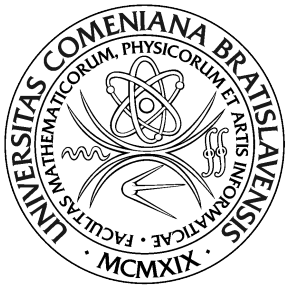
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



VIZUÁLNY SYSTÉM PRE INTERAKCIU ĽUDSKÉHO UČITEĽA S HUMANOIDNÝM ROBOTOM

Diplomová práca

2022 Bc. Nicolas Orság

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

VIZUÁLNY SYSTÉM PRE INTERAKCIU ĽUDSKÉHO UČITEĽA S HUMANOIDNÝM ROBOTOM

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Viktor Kocur, PhD.

Bratislava, 2022 Bc. Nicolas Orság

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Nicolas Orság
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vizuálny systém pre interakciu ľudského učiteľa s humanoidným robotom
Visual system for interaction of a human teacher with a humanoid robot

Anotácia: Toto zadanie je súčasťou projektu interakcie ľudského učiteľa s robotom. Robot pri tejto interakcii manipuluje jednoduchými objektmi na základe pokynov od ľudského učiteľa. Pre tento účel je tak vhodné aby robot dokázal správne detegovať pozíciu jednoduchých, objektov, učiteľovú a svoje ruky. Následne je potrebné aby robot dokázal rozpoznať zadané inštrukcie.

Cieľ: Cieľom tejto práce je navrhnúť, implementovať a otestovať systém ktorý na základe vstupných stereo dát z kamier v robotovi NICO deteguje pozíciu jednoduchých objektov, učiteľovej ruky a robotových rúk a následne rozpozná gestá od učiteľa. Súčasťou práce bude prehľad existujúcich riešení detekcie objektov v stereo snímkach a rozpoznávaní gest učiteľa. Navrhnutý systém bude vyhodnotený v kontexte prebiehajúceho projektu interakcie ľudského učiteľa s robotom.

Vedúci: Ing. Viktor Kocur, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 05.10.2021

Dátum schválenia: 06.10.2021

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

. . . . . .

Bratislava, 2022 Bc. Nicolas Orság

Poďakovanie

Abstrakt

Oblasť interakcie človeka a robota zaznamenala v posledných rokoch výrazný pokrok, pričom humanoidné roboty sa vyvíjajú pre široké spektrum aplikácií. Efektívna interakcia medzi týmito robotmi a ich používateľmi je však stále výzvou. Táto práca predstavuje vizuálny systém interakcie medzi ľudským učiteľom a humanoidným robotom. Systém využíva few-shot object detection algoritmy na učenie sa a rozpoznavanie novych objektov a gest.

Kľúčové slová: Počítačové videnie, Humanoidný robot, Hlboké učenie, Konvolučné neurónové siete, Few-shot object detection

Abstract

The field of human-robot interaction has seen significant progress in recent years, with humanoid robots being developed for a wide range of applications. However, effective interaction between these robots and their human users is still a challenge. This thesis presents a visual system for interaction between a human teacher and a humanoid robot. The system uses few-shot object detection algorithms for learning and recognising new objects and gestures.

Keywords: Computer vision, Humanoid robot, Deep learning, Convolutional neural networks, Few-shot object detection

Obsah

Kapitola 1

Úvod

Interakcia medzi ľuďmi a robotmi sa stáva čoraz dôležitejšou oblasťou v posledných rokoch vďaka rozvoju robotiky a jej aplikácií v rôznych odvetviach. Avšak účinná interakcia medzi ľuďmi a robotmi stále predstavuje výzvu, najmä v úlohách, ktoré zahŕňajú prirodzenú komunikáciu a ľudské gestá.

Cieľom tejto práce je vyvinúť vizuálny systém, ktorý umožní humanoidným robotom pochopiť a reagovať na základné gestá ľudského učiteľa a učenie sa rozpoznávať nové objekty.

Táto práca sa bude snažiť prispieť k oblasti interakcie medzi ľuďmi a robotmi vyvinutím vizuálneho systému na rozpoznávanie gest a objektov, pomocou preskúmania a použitia aktuálnych prístupov few-shot object detection, ktorý by mohol zlepšiť interakciu medzi ľudskými učiteľmi a humanoidnými robotmi.

Kapitola 2

Motivácia

V posledných rokoch je rastúci záujem o používanie robotov v rôznych odvetviach ako napríklad školstvo, zdravotníctvo, zábava a priemyselná výroba. Avšak na to aby roboti vedeli plniť úlohy efektívne, musia byť schopný prirodzene interagovať s ľuďmi. Toto je náročné veľmi náročné pri humanoidných robotoch, ktorý by mali napodobovať ľudský vzhľad a správanie.

Cieľom tejto diplomovej práce je vytvorenie vizuálneho systému, ktorý umožní humanoidnému robotovi rozpoznať učiteľové gestá a objekty pomocou few-shot object detection algoritmov, ktoré sú založené na konvolučných neuronových sieťach a hlbokom učení.

Kapitola 3

Úvod do problematiky

V úvodnej kapitole sa budeme venovať základným pojmom, ktoré sú nevyhnutné pre pochopenie nášho výskumu. Vysvetlíme si ako funguje počítačové videnie, objektová detekcia, konvolučné neuronové siete a taktiež sa zameriame na výskum objektovej detekcie pri malej trénovacej množine (Few shot object detection), ktorý budeme chcieť využiť v našej práci pre učenie nových objektov aj z veľmi malého množstva dát.

3.1 Počítačové videnie

V súčasnosti predstavuje počítačové videnie v informatike veľmi rýchlo rastúci a progresívny smer. Snaží sa priblížiť vnímaniu sveta z pohľadu ľudského oka, ktoré je pre nás prirodzené a automaticky sme schopný rozpoznávať objekty, farby a kontext toho čo vidíme. Avšak plné sémanticke pochopenie videnej reality je veľmi komplexné a zatiaľ nie sme schopný ho získať spracovaním digitálneho obrazu. Hlavne preto, že pochopenie obrazu môže vyplývať zo súvislostí, ktoré nie sú súčasťou obrazu.

Avšak počítačové videnie sa posúva veľmi rýchlo vpred. Neustále vyni-

kajú nové algoritmy a prístupy či už na detekciu objektov alebo klasifikáciu obrazu. Medzi základné problémy počítačového videnia patrí klasifikácia, objektová detekcia a segmentácia. Pri klasifikácii sa snažíme obraz priradiť do jednej z tried. V objektovej detekcii sa snažíme v obraze určiť oblasti všetkých známych objektov a priradiť ich do tried. A pri segmentácii je našim cieľom rozdeliť obraz do viacero oblastí a každému pixlu určiť oblasť do ktorej patrí.

3.2 Príznaky

Pri riešení problémov v počítačovom videní sa využívajú príznaky. Príznak v počítačovom videní je meratelný kus dát v obrázku, ktorý je unikátny pre špecifický objekt. Príznak môže reprezentovať napríklad štýl sfarbenia, nejaký tvar, či už čiaru alebo hranu v obraze alebo nejakú časť obrazu. Vďaka dobrému príznaku dokážeme od seba rozlíšiť objekty. Napríklad ak máme rozlíšiť mačku a bicykel tak ako dobrý príznak by mohlo byť, že na obrázku sa nachádza koleso. Hneď by sme vedeli vďaka tomuto príznaku klasifikovať obrázok do týchto dvoch tried. Ak by sme však mali za úlohu zistiť či je na obrázku motorka alebo bicykel, tak by nám tento príznak veľmi nepomohol a museli by sme pozerať na iné príznaky. Preto zväčša neextrahujeme z obrázku len jeden príznak, ale pre lepšiu detekciu vyberáme viacej príznakov, ktoré tvoria príznakový vektor.

Nie je presná definícia aké príznaky obrázku by sme mali použiť, ale závisí to skôr od nášho cieľu a typu úlohy. Príznaky sa delia na lokálne a globálne. Príznaky, ktoré platia pre celý obrázok, sa nazývajú globálne príznaky. Napríklad ako ako veľmi sú dominantné jednotlivé farby v obrázku. Globálny príznak nám opisuje obraz ako celok a mal by reporezentovať nejakú jeho špecifickú vlastnosť. Lokálne príznaky sa extrahujú len z určitej zaujímavej ob-

lasti v obrázku, využivajú sa najmä pri objektovej detekcii. Najskôr nájdeme zaujímavé oblasti, ktoré by mohli reprezentovať nejakú zaujímavú vlastnosť alebo nejaký objekt. Následne vytvoríme príznakový vektor pre danú oblasť, ktorý by nám mal poskytnúť zásadnú informáciu o tejto časti obrazu. Treba rátať s tým, že objekt na obrázku môže byť rôznej veľkosti, rôzne natočený, rôzne osvetlený, zašumený, môže sa nachádzať v rôznych častiach obrázku a podobne. Preto naše príznaky by mali byť ideálne invariantné voči týmto zmenám.

3.3 Objektová detekcia

Asi najskúmanejším problémom v počítačovom videní je objektová detekcia, ktorá spočíva v rozpoznaní jednotlivých objektov a ich pozícii v digitálnom obraze. K tomuto problému sa dá pristupovať tradičnými metódami počítačového videnia, alebo dnes už veľmi rozšireným s oveľa lepšími a presnejšími výsledkami, ako pri tradičných metódach a to pomocou hlbokého učenia, ktorých klúčom je naučiť sa na veľkých dátach extrahovať príznaky tak aby mala detekcia čo najväčšiu presnosť.

3.3.1 Tradičé metódy

Ako prvé vznikli tradičné metódy. Vysvetlíme si ako fungujú , pretože nám to pomôže pochopiť ako funguje dnes najvýuživanejší a najpresnejší prístup hlbokého učenia na ktorý sa zameriame v tejto práci. Tradičné metódy v objektovej detekcii majú zvyčajne tri etapy: vybratie oblasti, extrakcia príznakov, klasifikácia objektu.

V prvej etape sa snažíme lokalizovať objekt. Keďže objekt môže byť rôznej veľkosti, musíme skenovať celý obrázok pomocou posúvneho okna rôznej

veľkosti. Táto metóda je výpočtovo náročná.

V druhej etape použijeme použijeme metódy ako SIFT [1], HOG [2] na extrakciu vizuálnych príznakov na rozpoznanie objektu. Tieto príznaky nám poskytujú sémantickú a robustnú reprezentáciu. Avšak kvôli rôznemu osvetleniu, pozadiu a ulhu pohľadu je veľmi náročné manuálne navrhnúť deskriptor príznakov, ktorý by dokonale opísal všetky typy objektov.

V treťej fáze klasifikácie objektu používame zväčša Support Vector Machine(SVM) [3] alebo Adaboost [4] pre klasifikáciu cieľových objektov zo všetkých kategórii aby bola reprezentácia viac hierarchická, sémantická a informatívnejšia pre vizuálne rozpoznávanie.

Problémom pri tradičných metódach je výpočtová náročnosť pri generovaní kandidátov na bouning box (obdĺžnik ohraničujúci objekt) pomocou techniky posúvneho okna a taktiež manuálne nastavenie extrakcie príznakov nie je vždy veľmi presné. Avšak ich výhodou je, že nepotrebujeme veľký anotovaný dataset a taktiež veľkú výpočtovú silu pri tréningu, ktoré potrebujeme pri prístupe hlbokého učenia.

3.3.2 Metódy hlbokého učenia

Neskôr keď tradičné metódy začali stagnovať, sa začali na riešenie problémov klasifikácie obrázkov, objektovej detekcie a segmentácie využívať metódy hlbokého učenia. Hlavným dôvodom, prečo metódy hlbokého učenia dosahujú lepšie výsledky ako tradičné metódy je, že netreba manuálne voliť príznaky, ale ich úlohou je nájsť najlepšie príznaky pre danú úlohu. Využívajú na to neurónové siete.

Neurónové siete

Základným stavebným blokom neurónovej siete je neurón. Vysvetlíme si

ako funguje neurónova sieť zložená len z jedného neurónu. Do neurónu vstupuje m vstupov ktoré predstavuje vstupný vektor $\vec{x} = (x_1, x_2, ...x_m)$. Každý z týchto vstupov má svoju váhu, váhy reprezunetujeme váhovým vektorom $\vec{w} = (w_1, w_2, ...w_m)$. Najprv spravíme skalárný súčin vstupného a váhového vektora a prirátame k tomu bias b. Následne aplikujeme do nelienárnej aktivačnej funkcie f a dostaneme výstup z neurónu. Hlavnou úlohou aktivačnej funkcie je zavedenie nelinearity.

$$y = f(\vec{w} \cdot \vec{x} + b) \tag{3.1}$$

Neurónová sieť sa skladá zväčša z viacej neurónov a viacerých vrstiev. Výstup neurónu z nižšej vrstvy môže byť vstupom do neurónu vo vyššej vrstve a ako sme si popísali vyššie má svoju váhu, ktorá popisuje silu spojenia medzi dvoma neurónmi.

Váhy $\vec{w}$ a bias b sú parametre, ktoré sa pri tréningu neurónovej siete prispôsobujú, tak aby sa minimalizoval rozdiel medzi výstupom siete a očakávaným výstupom.

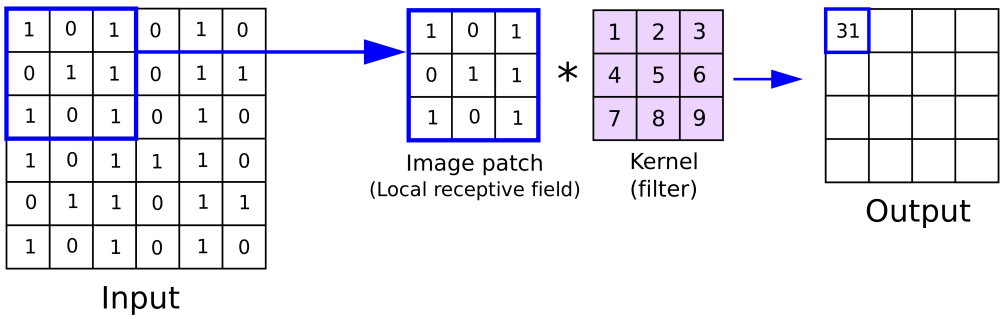
Najviac využívaným typom neurónových sietí v počítačovom videní sú konvolučné neuronové siete, ktoré sú ideálne na spracovanie dát v mriežkovitom tvare ako napríklad obrázkov. O nich si povieme viac v ďalšej časti.

Úvod do CNN

Konvolučné neuronové siete (CNN) [5] sú typ neurónovej siete, ktorá obsahuje konvolučné vrstvy. Konvolučné vrstvy skenujú dáta pomocou množiny filtrov, kde každý filter hľadá špecifický vzor v dátach. Vďaka týmto vrstvám sú CNN ideálne na spracovanie dát mriežkovitého tvaru ako napríklad obrázky. Teraz si vysvetlíme ako fungujú kovolučné vrstvy.

Konvolučné vrstvy

Ako sme si spomenuli v úvode, konvolúcia sa deje pomocou filtrov. Aplikáciu filtra si vysvetlíme pomocou konkrétneho príkladu na obrázku 3.1 kde máme vstupný obrázok veľkosti 6x6 a filter veľkosti 3x3, po aplikácii tohto filtru na obrázok dostaneme výstup veľkosti 4x4. Na obrázku 3.1 vidíme vyrátanú prvú hodnotu na výstupe. Túto hodnotu dostaneme prenásobením hodnôt ľavého horného rohu veľkosti filtra z obrazu (modré okno na obrázku) s hodnotami filtra a následným sčítaním násobkov. Pre vyrátanie druhej hodnoty v prvom riadku sa naše modré okno veľkosti 3x3, ktoré predstavuje hodnoty brané z obrázka posunie o 1 doprava a aplikujeme rovnaký postup ako pri výpočte prvej výstupnej hodnoty s novými hodnotami z obrázka. Takýmto spôsobom vyrátame všetky hodnoty v prvom riadku a následne pre ďalší riadok posunieme naše okno, ktoré berie hodnoty z obrázka o 1 nadol.

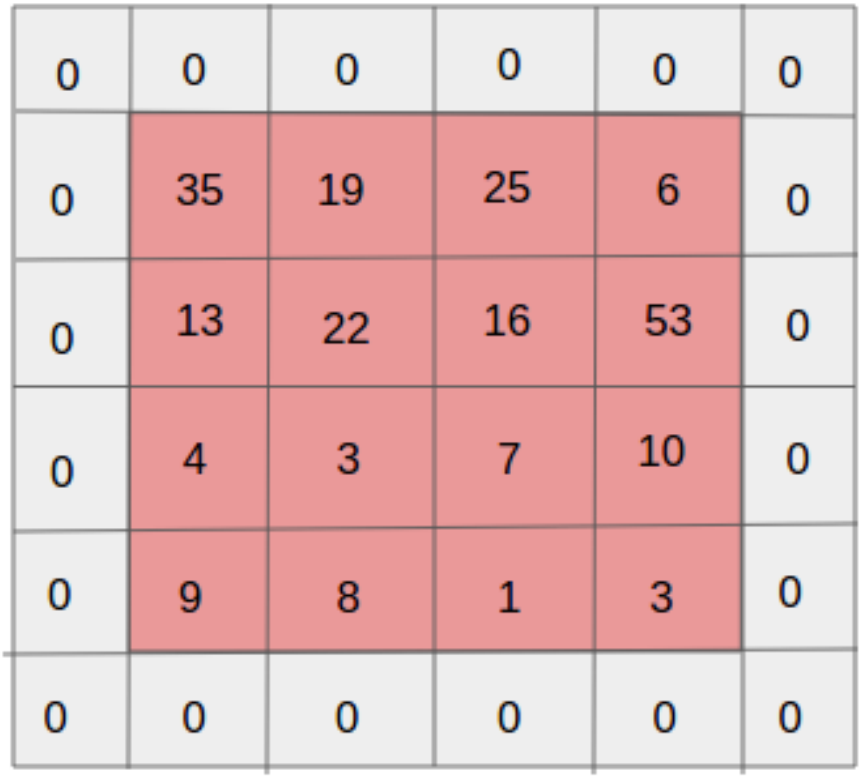


Obr. 3.1: Aplikácia filtra

Toto posunutie sa nazýva stride a dá sa nastaviť aj na vyššie číslo ako 1.

Ďalší nastaviteľný parameter pri konvolúcii je padding kde zväčšíme veľkosť nášho vstupu tak, že pridáme rám okolo nášho vstupu s nulovými hodnotami ako vidíme na obrázku 3.2 bielou farbou pridané hodnoty pri paddingu

veľkosti 1. Veľkosť paddingu predstavuje šírku tohto rámu. Padding by nemal byť väčší ako výška alebo šírka filtra. Vďaka paddingu vieme napríklad dosiahnúť, že rozmery výstupu budú rovnaké ako rozmery vstupu, pri rôznych rozmeroch filtra.



Obr. 3.2: Znázornenie paddingu pri konvolúcii

Pri nastavovaní rozmerov filtra, stridu a paddingu, musí ich kombinácia sedieť na rozmery vstupu, tak aby sa dal celý vstup prejsť filtrom.

Vstupný obrázok môže mať viacero kanálov, zväčša má 3 kanály, každý pre jednu z farieb červená, modrá a zelená, teda tvorí ho tenzor tretieho rádu. Filter teda bude tiež tenzor tretieho rádu a jeho posledná dimenzia

bude rovnako veľká ako posledná dimenzia vstupu. Jeden filter nám teda vráti tenzor tretieho rádu s veľkosťou poslednej dimenzie 1. Napríklad máme vstupný obraz veľkosti 32x32x3, padding = 0, stride = 0 a filter veľkosti 5x5x3, po aplikácií filtra dostaneme výstup veľkosti 28x28x1.

Filtrov môže byť pri konvolúcii kludne aj viac ako 1. Počet filtrov nám nastavuje veľkosť poslednej dimenzie výstupu. Napríklad pri vstupe 6x6x3 a použití 5 tich filtrov veľkosti 3x3x3 stride = 1 a padding = 1 dostaneme výstup s rozmermi 6x6x5. Pri ďalšej konvolučnej vrstve bude musieť byť posledná dimenzia veľkosti filtrov = 5. V ďalšej časti si predstavíme pooling.

Pooling

Po konvolučnej vrstve sa v CNN zvyčajne nachádza poolingová vrstva, ktorá zvyčajne zmenšuje priestorovú dimenziu. Poznáme niekoľko typov poolingu, vrátane max poolingu a average poolingu, ale najbežnejším je max pooling, ktorý vezme maximálnu hodnotu každého poolingového regiónu.

Napríklad, ak pooling región je mriežka 2x2, max pooling vezme najväčšiu zo 4roch hodnôt v mriežke a vráti ju ako jednu hodnotu do výstupnej príznakovej mapy. Toto spôsobí redukciu veľkosti príznakovej mapy a ponecháme len najdôležitejšie príznaky.

Plne prepojené vrstvy

Ďalej po konvolučných a poolingových vrstvách, CNN zvyčajne obsahujú jednu alebo viac plne prepojených vrstiev, ktoré kombinujú príznaky extrahované konvolučnými a poolingovými vrstvami na určenie finálnej predikcie.

Plne prepojená vrstva je zvyčajne výstupná vrstva, ktorá vracia finálnu predikciu CNN. Počet neurónov vo výstupnej vrstve zavisí od úlohy, ktorú

máme. Napríklad, pre klasifikáciu obrázku, výstupná vrstva môže mať jeden neurón pre každú triedu, a neurón s najvyššou aktivačnou hodnotou by predstavoval predikovanú triedu.

Tréning CNN

Tréning CNN zahŕňa prispôsobenie váh filtrov a spojení v sieti na minimalizovanie stratovej funkcie, ktorá meria rozdiel medzi predikciou a skutočným labelom. Proces trénovania CNN môže byť rozdelený na nasledovné kroky:

Prvým krokom je vyzbierať anotovaný dataset, ktorý bude použitý na trénovanie modelu. Tento dataset by mal byť dostatočne veľký a rôznorodý aby sa naša sieť vedela generalizovať na nové obrázky. Pred tréningom siete, je zväčša nevyhnutné predspracovanie dát, aby sme sa uistili, že sú vhodné pre CNN. To môže zahŕňať prispôsobenie veľkosti obrázkov alebo augmentáciu dát aplikovaním náhodných transformácii, pre rôznorodosť datasetu a predchádzaniu overfittingu.

Ďalším krokom je rozdelenie datasetu na treningovú, validačnú a testovaciu množinu. Trénovacia množina je použitá na trénovanie CNN, validačná na vyhodnotenie CNN počas tréningu a testovacia na vyhodnotenie modelu po tréningu. Validačná množina je nápomocná pre nastavenie hyperparametrov CNN. Hyperparametre sú nastaviteľné parametre, ktoré určujú nastavenie samotného trénovacieho procesu. Testovacia množina nám poskytuje približnú schopnosť generalizácie našej siete.

Tretím krokom je návrh CNN a voľba hyperparametrov. Je veľa spôsobov ako navrhnúť CNN, ktoré majú vplyv na jej výkon, avšak lepšie ako navrhovať vlastnú CNN je zobrať nejakú existujúcu CNN, ktorá dosahuje dobré výsledky na bežných benchmarkoch. Po zvolení CNN, je dôležité správne na-

stavenie hyperparametrov, pre nájdenie najlepšej kobinácie pre danú úlohu. Dobré je sieť skúšať trénovať a pomaly upravovať hyperparametre.

Ďalším krokom je určenie stratovej funkcie a optimalizačného algoritmu. Stratová funkcia meria ako dobre je náš model schopný predikovať žiadaný výstup pre konkrétny vstup a jej voľba záleží na type úlohy.

Optimalizačný algoritmus(optimalizátor) je zodpovendný za prispôsobenie parametrov siete na minimalizovanie stratovej funkcie. Najbežnejší optimaizačný algoritmus je stochastic gradient descent (SGD). Iný často používaný optimalizačný algoritmus je napríklad Adam [6].

Gradient Descent(Gradientový zostup) prispôsobuje parametre našej siete podľa vzorca 3.2. Vypočítaním gradientu stratovej funkcie $\frac{\partial L}{\partial w}$ získame smer k maximu stratovej funkcie v danom bode, teda s našimi aktualnými parametrami. My sa snažíme dosiahnúť minimum preto naše parametre upravíme v opačnom smere gradientu vynásobený krokom učenia(learning rate) η. Learning rate je jeden z hyperparametrov, ktorý predstavuje veľkosť kroku, ktorý spraví optimalizátor na upravenie parametrov siete. Tento optimalizačný krok iteratívne opakujeme, kým nám klesá validačná chyba. Gradient descent prispôsobuje parametre podľa všetkých tréningových príkladov.

$$w_{n+1} = w_n - \eta \frac{\partial L}{\partial w} \tag{3.2}$$

Stochastic gradient descent sa líši od gradient descentu tým, že neupravuje parametre vzhľadom na všetky tréningové príklady, ale len vzhľadom na niekoľko tréningových príkladov, počet týchto tréningových príkladov závisí od ďaľšieho hyperparametra: veľkost batchu. Veľkosť batchu je počet tréningových príkladov použitých v jednej iterácii tréningu. SGD je rýchlejšie a menej výpočtovo náročné, taktiež sa pridáva šum do optimalizačného procesu, čo znižuje šancu, že algoritmus skončí v lokálnom minime.

Po trénovaní CNN je dôležité vyhodnotiť výkon na testovacej množine. Testovacia množina by mala byť dostatočne veľká aby nám ponkla spolahlivé vyhodnotenie nášho modelu a nemala by byť použitá pri tréningu.

Výkon CNN môže byť vyhodnotený pomocou rôznych metrík. Zavisí od úlohy, ktorú metriku je pre nás vhodné použiť.

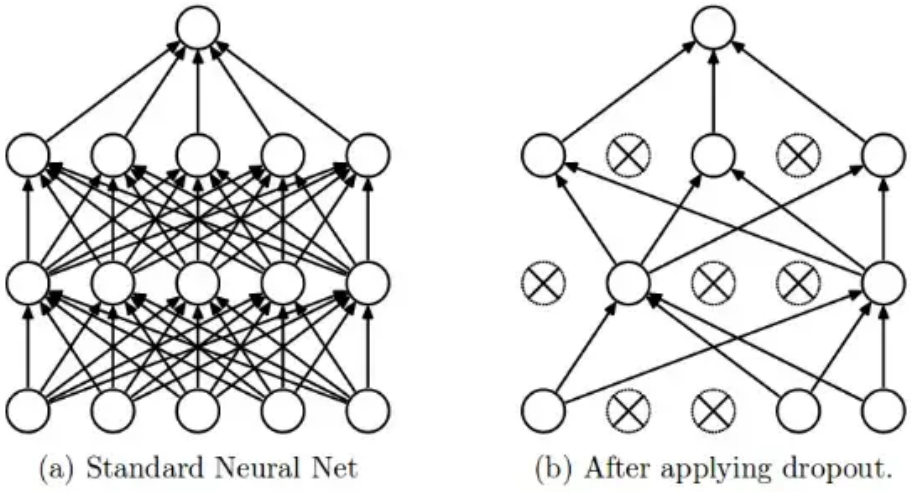
Aktivačné funkcie a regularizácia

Za účelom zavedenia nelinearity do siete zvyčajne CNN obsahujú aktivačné funkcie v konvolučných a plne prepojených vrstvách. Najbežnejšou aktivačnou funkciou je Rectified Linear Unit (ReLU), ktorá má formu $f(x) = \max(0, x)$. Používajú sa aj iné aktivačné funkcie ako napríklad sigmoid a tanh.

Overfitting je pretrénovanie siete, to znamená, že náša sieť funguje príliš dobre na tréningových dátach, ale má to negatívny vplyv na výkon siete na nových dátach, ktoré neboli videné pri tréningu. Teda celkový výkon našej siete všeobecne na všetkých dátach sa overfittingom znižuje.

Na prevenciu overfittingu a zlepšenie generalizácie, konvolučné neurónové siete využívajú regularizačné techniky ako napríklad dropout.

Dropout náhodne dropne(nepoužije) nejaké percento neurónov v sieti počas každej tréningovej iterácie. ako vidíme na obrázku 3.3



Obr. 3.3: Aplikácia dropoutu

Benchmarkové metriky v objektovej detekcii

Na to aby sme vedeli vyhodnotiť presnosť modelov objektovej detekcie a porovnať ich medzi sebov potrebujeme používať rovnaké metriky. Najčastejšie používané benchmarkové metriky na určenie presnosti modelu objektovej detekcie sú mAP, mAP50, mAP75.

V objektovej detekcii, určujeme polohu objektu pomocou bounding boxu (obdĺžnik). A každému bounding boxu priradíme triedu, teda názov objektu v danej lokalite.

Pre vyhodnotenie porovnávame predikciu nášho modelu s manuálne anotovaným obrázkom. Najprv musíme spočítať počet true positive, false positive a false negative pre každý objekt zvlášť. True positive je správne detegovaný objekt. False positive je ak sme predikovali objekt tam kde nemal byť. False negative je keď sme nedetegovali objekt tam kde mal byť. Pomocou týchto hodnôt vyrátame precision 3.3 a recall 3.4

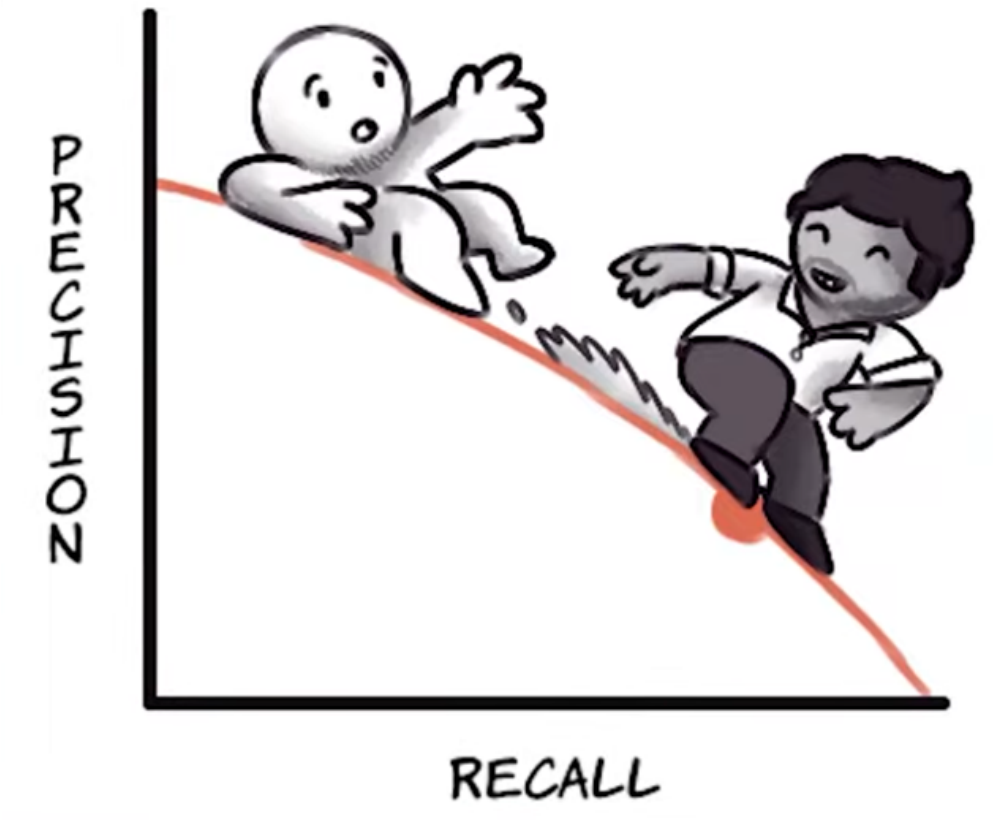
$$Precision = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} \tag{3.3}$$

$$Recall = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \tag{3.4}$$

Na určenie, či predikovaný bouning box je dostatočné presný sa používa metrika IoU(intersection over union). Teda plocha prieniku predikcie a anotácie deleno plocha zjednotenia prieniku a anotácie 3.5. Podľa hodnoty IoU threshold určujeme, či je predikcia true positive alebo false positive. Ak je IoU >= IoU threshold pri konkrétnej preidikcii je to true positive inak je to false positive.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{3.5}$$

AP50 je precission * recall pri hodnote IoU threshold = 0.5. AP75 je precission * recall pri hodnote IoU threshold = 0.75. Pre výpočet AP musíme vyrátať precision a recall pre rôzne hodnoty IoU threshold. Následne zostrojíme precision recall krivku 3.4. Obsah pod touto krivkou je naše AP.



Obr. 3.4: Krivka precision recall

Pre vyrátanie mAP musíme vyrátať AP pre každú triedu a následne spraviť priemer. Rovnako pre mAP50 a mAP75.

Faster R-CNN

Jedna z najpoúživanejších metód objektovej detekcie je Faster R-CNN [7]. Skladá sa z 3 hlavných častí: Backbone, RPN(Region Proposal Network) a RoI Heads.

Backbone

Backbone je kľúčovou časťou siete, ktorá zabezpečuje extrakciu príznakov zo vstupného obrázka. Faster R-CNN je pružná sieť, ktorá využíva existujúcu sieťovú architektúru, ako napríklad VGG [8], ResNet [9] alebo Inception, ako backbone na extrakciu príznakov.

RPN

Výstup z backbone je príznaková mapa, ktorá je vstupom pre RPN. RPN má za úlohu vygenerovať návrhy regiónov, ktoré pravdepodobne obsahujú objekt. Toto RPN dosiahne pomocou pomocou prístupu posúvneho okna a anchor boxov.

1. Anchor boxy: RPN používa množinu preddefinovaných anochor boxov. Anchor boxy sú bounding boxy s pevne stanovenou veľkosťou, ktoré sú husto rozmiestnené po príznakovej mape. Tieto anchor boxy pokrývajú rôzne škály a pomer strán, aby zachytili objekty rôznych veľkostí a tvarov.

2. Prístup s posúvnym oknom: RPN posúva malé okno zväčša 3x3 alebo 1x1 po príznakovej mape, centruje ho v každom bode príznakovej mapy a používa anchor boxy na generovanie návrhov regiónov. Pre každý anchor box v danej lokalite RPN predikuje dve veci:

   (a) Objectness score: RPN predikuje pravdepodobnosť, že anchor box je objekt alebo pozadie. To sa dosiahne použitím binary classification head, ktorej výstup je skóre objektu pre každý anchor box.

   (b) Bounding Box Regression: RPN taktiež predikuje súradnice bounding boxu, ktorý tesne obklopuje objekt, ak existuje, spojeného s

anchor boxom. To sa dosiahne pomocou bounding box regression head, ktorej výstup sú jemne doladené súradnice návrhu bounding boxu na základe odchýlok od súradníc anchor boxu.

3. Non-Maximum Suppression (NMS): Po získaní Objectness score a Bounding Box Regression predikcií pre všetky anchor boxy RPN aplikuje postup Non-Maximum Suppression (NMS), aby odfiltroval nadbytočné návrhy. NMS je postup, ktorý odstraňuje prekrývajúce sa návrhy, ponecháva iba návrhy s najvyšším Objectness score a odstraňuje duplikáty alebo nadbytočné návrhy.

4. Doladenie návrhov: zvyšné návrhy regiónov sú doladené aplikovaním predikovanej bounding box regresie na anchor boxy.

5. Návrhy regiónov: Výstupom RPN je množina regiónov, pričom každý región je charekterizovaný súradnicami, výškou a širkou, taktiež má priradené skóre objektu(Objectness score).

RoI Heads

Roi Heads sa skladaju z troch častí: RoI Pooler, Box Head a Box Predictor. Vstupom pre RoI Heads je príznaková mapa(výstup z backbone) a taktiež návrhy regiónov(výstup z RPN).

1. RoI Pooler: Je zodpovedný za zmenšenie rôznych veľkostí a tvarov RoI na jednotnú veľkosť, ktorá je vhodná pre vstup do ďalších častí modelu. RoI(Region of Interest) sú návrhy regiónov, ktoré nám vygenerovala RPN. Vezme návrhy regiónov, ktoré sú rôznych veľkostí a tvarov a pomocou RoI Allign ich transformuje na rovnakú veľkosť, zväčša je to 7x7 alebo 14x14. RoI Aliign je presnejšia alternatíva ako max pooling, ktorá umožňuje interpoláciu hodnôt príznakov na subpixlovú presnosť,

čo zlepšuje presnosť zarovnania RoIs na príznakovú mapu. To znamená, že RoI Align dokáže zarovnať RoIs na hodnoty medzi pixelmi na príznakovej mape.

2. Box Head: Po RoI Alligne fixné veľkosti reprezentácií príznakov regiónov prechádzajú cez Box Head, ktorá pozostáva z jedného alebo viacerých plne prepojených (FC) vrstiev nasledovaných aktivačnými funkciami. Box Head je zodpovedný za ďalšie doladenie reprezentácií príznakov a generovanie predikcií pre triedy objektov a presnejšie súradnice bounding boxov.

3. Box Predictor: Výstup z Box Head ide do Box Predicotra, ktorý je zodpovedný za predikciu triedy objektu a upravenie súradníc bounding boxu, skladá sa z dvoch častí:

   (a) Box Classifier: Táto časť je zodpovedná za predikciu triedy objektu pre každý RoI. Zvyčajne sa skladá z jedného alebo viacerých plne prepojených (FC) vrstiev, nasledovaných aktivačnou funkciou softmax, ktorá generuje pravdepodobnosti pre rôzne triedy objektov

   (b) Box Regressor: Táto časť je zodpovedná za predikciu presnejších hodnôt bounding boxu pre každý RoI. Zvyčajne sa skladá z jedného alebo viacerých plne prepojených (FC) vrstiev, nasledovaných aktivačnými funkciami ako ReLU alebo sigmoid, ktoré generujú predpovedané posuny pre bounding boxy.

3.4 Few-shot object detection(FSOD)

Problémom pri väčšine algoritmov objektovej detekcie je, že vyžadujú

veľký dataset anotovaných obrázkov na tréning modelu, čo môže byť drahé a časovo náročné.

Few-shot object detection je varianta objektovej detekcie, ktorá sa snaží učiť z malého datasetu. Je to inšpirované few-shot learningom, čo je typ strojového učenia, ktorý sa učí na malých trénovacích dátach. Few-shot learning si získal v posledných rokoch veľa pozornosti, vďaka jeho schopnosti adaptovať sa novým taskom s malým množstvom dát, čo je dôležité v prípade, že nemáme dostatok anotovaných dát.

Few-shot object detection je náročný problém, pretože model sa musí naučiť chrakteristiky objektov z malého množstva príkladov, čo je náročné vzhľadom na komplexnosť a rôznorodosť objektov. Naviac model musí rozpoznať nové triedy, ktoré neboli videné počas tréningu, čo vyžaduje dobré rozlišovanie medzi odlišnými triedami.

Napriek náročnosti, je to dôležitý problém, pretože má potenciál výrazne znížiť počet anotovaných dát potrebných na objektovú detekciu.

3.4.1 Prístupy k FSOD

Sú viaceré prístupy, ktoré boli navrhnuté na FSOD, môžu byť zhrnuté do troch hlavných kategórií: meta-learning, transfer learning a augmentácia dát.

Meta-learning

Meta-learning je prístup, ktorý sa snaží naučiť sa učiť. Sústredí sa na trénovanie modelu, ktorý sa vie rýchlo prispôsobiť novým úlohám iba vďaka veľmi malému počtu obrázkov. Tento prístup zvyčajne zahŕňa trénovanie modelu, ktorý sa vie učiť z malého počtu dát buď použitím vonkajšej pamäti alebo optimalizačného algoritmu. Napríklad Model-Agnostic Meta-Learning

(MAML) [10] algoritmus používa gradientový optimalizačný algoritmus na tréning modelu, ktorý sa vie prispôsobiť novým úlohám malým počtom aktualizácií gradientu. Algoritmus MAML bol aplikovaný na few-shot object detection pri fine-tuningu predtrénovaného modelu na objektovú detekciu na malom počte obrázkov.

Transfer learning

Transfer learning je technika pri ktorej sa využívajú parametre natrénovanej siete z jednej úlohy ako iniciálne parametre pre sieť na novú veľmi podobnú úlohu. Napríklad sieť trénovaná na veľkom datasete obrázkov zvierat by mohla byť použitá ako iniciaálna sieť pre trénovanie siete na rozpoznávanie špecifického typu zvierat ako napríklad plemená psov. Použitím siete s predtrenovanými váhami, sa naša sieť naučí rýchlejšie rozpoznávať plemená psov a stačí na to menej dát.

Augmentácia dát

Prístup augmentácie dát spočíva v rozmnožení malého množstva dát pomocou aplikovania rôznych transformácií. Zvýšením počtu dát, sa model môže naučiť robustnejšie príznaky. Pri augmentácii sa používajú transformácie ako rotácia, škálovanie, zašumenie. Používame transformácie obrazu, ktoré menia obraz, ale nemenia jeho sémantický obsah.

3.4.2 Datasety pre vyhodnotenie FSOD

Na vyhodnotenie výkonu FSOD algoritmov, výskumnici používajú verejne dostupné datasety a vyhodnocovacie metriky. Tieto datasety a metriky slúžia na porovnanie výkonu odlišných algoritmov a ich všeobecnosti. Najpoužívanejšie datasety sú:

COCO dataset [11], ktorý obsahuje 80 tried a viac ako 330 000 anotovaných obrázkov. VOC dataset [12], je to populárny dataset, ktorý obsahuje 20 tried a viac ako 11 000 obrázkov.

Tieto datasety sú používané ako štandard pre few-shot object detection, vyberie sa z nich niekoľko tried, ktoré sa považuju ako novel classes (triedy s malym počtom anotovaných dát), pre tieto striedy sa použije iba zopár anotovaných obrázkov(few-shot) a zvyšné triedy sa použijú na predtrénovanie modelu.

3.4.3 Aktuálne riešenia FSOD

K FSOD bolo publikovaných viacero článkov, a veľa autorov použilo iné datasety a spôsoby vyhodnotenia ich modely. Preto je náročné ich porovnanie. Avšak, vo všeobecnosti meta-learningové prístupy zvyknú dosahovať lepšie výsledky ako transfer learning alebo prístup augmentácie dát. Hlavne preto, že meta-learningové prístupy sú špeciálne navrhnuté učiť sa z malého počtu príkladov.

Avšak, je potrebné zmieniť, že výkon few-shot object detection modelu veľmi závisí od konkrétnej implementácie, zvolených dát a zvolených metrík. A taktiež treba brať do úvahy výpočtovú a pamäťovú náročnosť.

Frustrantingly simple few-shot object detection

Frustrantingly simple few-shot object detection [13] je metóda, ktorú sme sa rozhodli použiť v tejto práci. Kľúčová myšlienka za touto metódou je naučiť sa detegovať objekty tréningom na množine základných tried (base classes) s veľkým počtom anotovaných obrázkov a následne spraviť fine-tuning detektora na malom množstve anotovaných obrázkov z nových tried(novel classes).

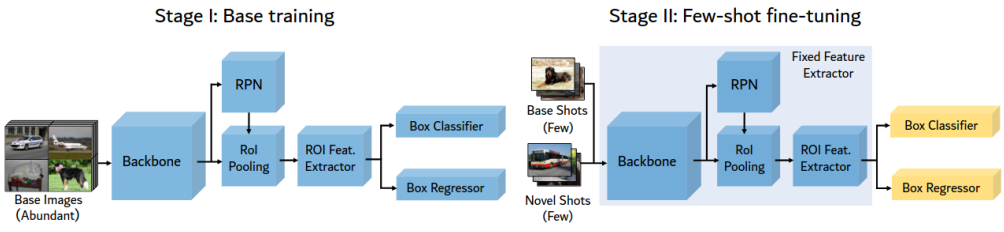
Využíva metódu Faster R-CNN, ktorú sme si popísali v predošlej kapitole. Spočíva v tom, že rozdelíme náš tréning na 2 etapy. V prvej etape sa vykoná base tréning na base classes klasicky cez Faster R-CNN pomocou stratovej funkcie 3.6.

$$\mathcal{L} = \mathcal{L}_{rpn} + \mathcal{L}_{cls} + \mathcal{L}_{loc} \tag{3.6}$$

Kde $\mathcal{L}_{rpn}$ sa aplikuje na výstup z RPN na rozlíšenie popredia od pozadia, $\mathcal{L}_{cls}$ je cross-entropy loss pre Box Classifier a $\mathcal{L}_{loc}$ je stratová funkcia pre Box Regressor.

Následne v druhej etape tréningu(few-shot fine-tuning) vytvoríme pre každú z tried(novel aj base) malý tréningový set. Pre novel classes priradíme náhodné inicializované váhy do siete pre Box Predictor. A následne robíme fine-tuning, ale len na Box Predictore, poslednej vrstve nášho modelu. Zvyšok siete ostáva zmrazený. Použijeme rovnakú loss funkciu a 20x nižší learning rate.

Na obrázku 3.5 vidíme znázornené tieto etapy a v druhej etape vidíme žltou farbou znázornený Box Predictor, jediná časť siete, ktorá počas fine-tuningu nie je zmrazená.



Obr. 3.5: Model pre frustrantingly simple few shot object detector

Kľučovým prvkom tejto metódy je oddelenie učenia sa reprezentácii príznakov a učenia sa predikovania boxov. Keďže príznaky, ktoré sme sa naučili

používať na base classes môžme využiť pre novell classes.

Kapitola 4

Testovanie FSFSODT

V predošlej kapitole sme si popísali ako funguje prístup metódy Frustratingly simple few shot object detection. Teraz si ju prakticky vyskúšame a otestujeme pre rôzny počet trénovacích obrázkov pre novel classes. A taktiež ako sa s ňou dá reálne pracovať a ako je rýchla posledná fáza tréningu fine-tuning pri trénovaní na jednom GPU konkrétne NVIDIA RTX 3070.

4.1 Dataset

Ako prvé, bolo potreba zvoliť a pripraviť dataset na ktorom budem túto metódu testovať. Zvolil som si veľmi známy dataset PASCAL VOC, ktorý sa bežne používa pre porovnanie presnosti medzi rôznymi algoritmami. Tento dataset obsahuje 20 tried.

Pri few-shot object detection, potrebujeme mať dataset rozdelený na triedy ku ktorým máme veľké množstvo anotovaných dát(base classes), a na triedy ku ktorým máme malé množstvo anotovaných dát(novel classes).

Ako base classes som použil triedy: aeroplane, bicycle, boat, bottle, car, cat, chair, diningtable, dog, horse, person, pottedplant, sheep, train, tvmo-

nitor.

Ako novel classes som použil triedy z pascal voc: bird, bus, cow, motorbike, sofa a taktiež som pridal jednu vlastnu triedu apple(anotované obrázky som stiahol z roboflow)

Keďže tvorcovia poskytli predtrénované modely, časovo a výpočtovo najnáročnejšiu prvú fázu tréningu base tréning som mohol vynechať, keďže používam rovnaké base classes dopadla by úplne rovnako a jej výstup môžem použiť pre následný fine tuning pre rôzne novel classes a aj pre rôzny počet anotovaných obrázkov pre novel classes.

Následne bolo treba pripraviť textové súbory pre každú triedu, ktoré obsahovali názvy obrázkov, ktoré budú použité pre fine-tuning. A teda v mojom prípade pri one-shot detekcii každý so súborov pre jednu triedu by mal obsahovať jeden anotovaný obrázok z môjho datasetu.

4.2 Tréning

Po stiahnutí predtrénovaného modelu na 15 base triedach bolo treba inicializovať hodnoty pre nové triedy v Box Classsifier. Box Classifier mal pôvodne výstup pravdepodobnostného rozloženia pre 15 tried, ale teraz máme 21 tried, takže treba váhy a biasy pre pôvodné triedy ponechať a inicializovať náhodné hodnoty pre nové triedy.

Nasleduje na rad fine-tuning. Keďže autori použili na tréning 8 16GB gpu, musel som prispôsobiť konfiguračné parametre tréningu aby som si vystačil s pamäťou 1 8GB gpu.

Znížil som batch size z 16 na 2, learning rate z 0.001 na 0.000125. A zvýšil som step z 3000 na 24000 a max_iter z 4000 na 32000. Teda batch size a base_learning rate(počiatočný learning rate) som 8-násobne znížil a

step (Počet iterácií, po ktorých sa learning rate zníži o desatinu) a max_iter (počet itérácií) som 8-násobne zvýšil. Pri každej k-shot detekcii máme odlišný step a max_iter. Čím viac trénovacích obrázkov, teda čím väčšie k tým je potrebné trénovať dlhšie pre optimálne výsledky a preto treba zvýšiť tieto dva parametre.

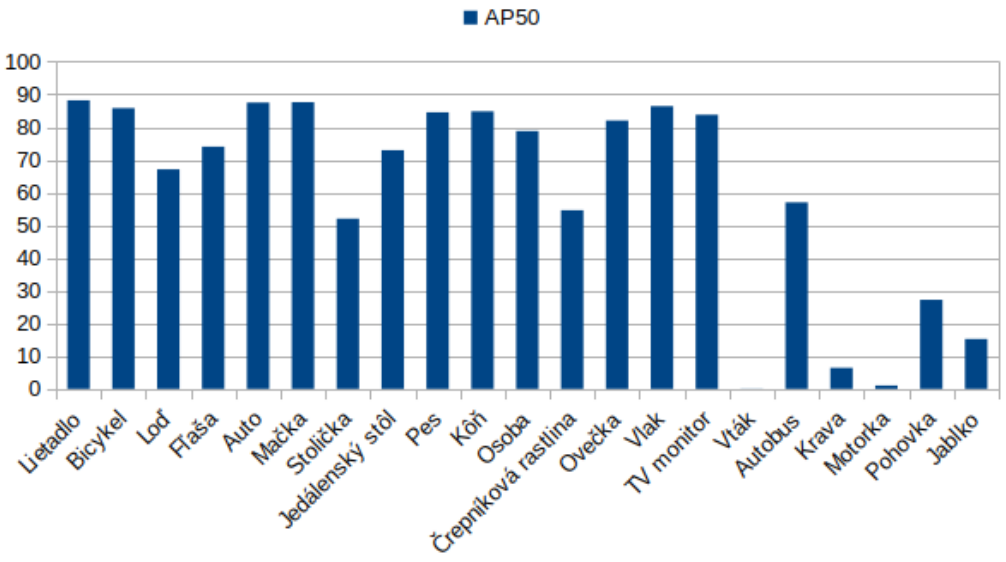
4.2.1 Testovanie rýchlosti a presnosti modelu pri zmene počtu trénovacích obrázkov novel tried

Teraz si otestujeme ako rýchly bude tréning a následne akú presnosť bude dosahovať model pri rôznom počte trénovacích obrázkov novel tried. Trénovacie obrázky pre každú triedu sú vybrané náhodne.

1-shot detekcia

Najprv som sa rozhodol algoritmus otestovať pre 1-shot detekciu, teda druhá fáza tréningu fine-tuning bude prebiehať len na 1 obrázku z každej triedy (novel aj base) teda na 21 obrázkoch, ktoré budú taktiež resizované tak že kratšia hrana obrázku bude resizovaná na tieto dĺžky: 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800 a dlhšia hrana bude prispôsobená tak aby bol zachovaný pomer hrán, stým že maximálna veľkosť dlhšej hrany je 1333 a teda ak by mal náš resize presiahnúť túto veľkosť nastaví sa dlhšia hrana obrázka na 1333 a kratšia hrana sa prispôsobí na veľkosť aby sa zachoval rovnaký pomer strán ako pri originálnom obrázku.

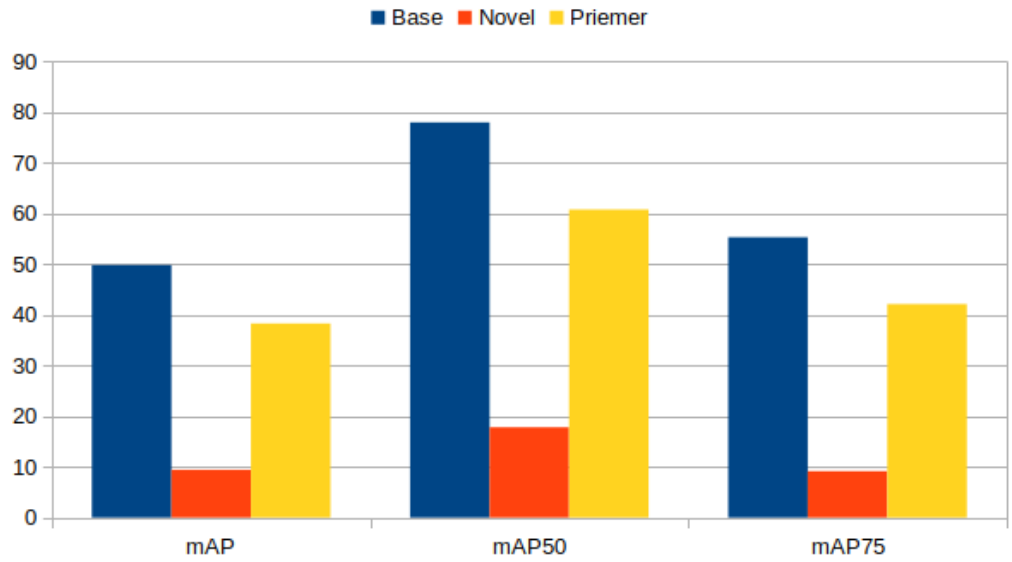
Celkový čas tréningu: 55 minút a 6 sekúnd, na obrázku 4.1 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti nie je rovnomerné a pre niektoré triedy je presnosť takmer nulová. Na obrázku 4.2 a 4.3 vidíme priemernú presnosť nášho modelu.



Obr. 4.1: AP50 pre jednotlivé triedy

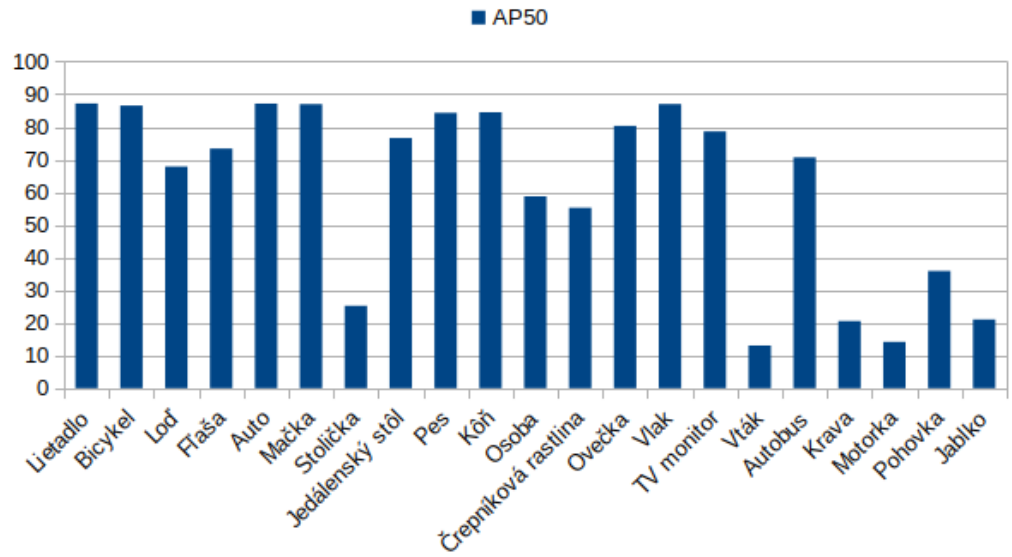
Presnosť	Base	Novel	Priemer
mAP	49.858	9.41	38.301
mAP50	78.004	17.812	60.806
mAP75	55.335	9.142	42.137

Obr. 4.2: Tabuľka presnosti pre všetky triedy



Obr. 4.3: Celková presnosť pre všetky triedy

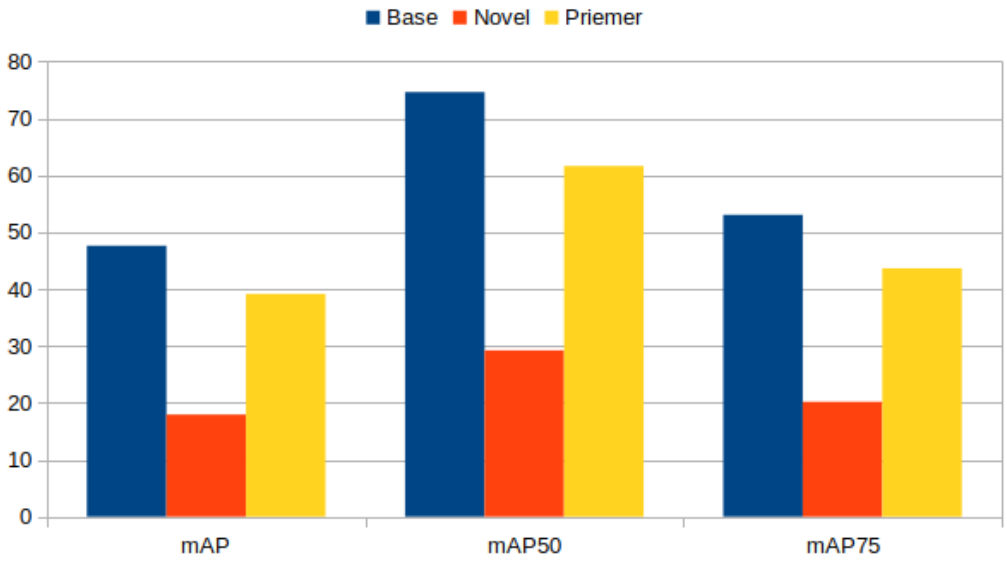
2-shot detekcia



Obr. 4.4: AP50 pre jednotlivé triedy

Presnosť	Base	Novel	Priemer
mAP	47.623	17.947	39.144
mAP50	74.623	29.218	61.65
mAP75	53.055	20.182	43.663

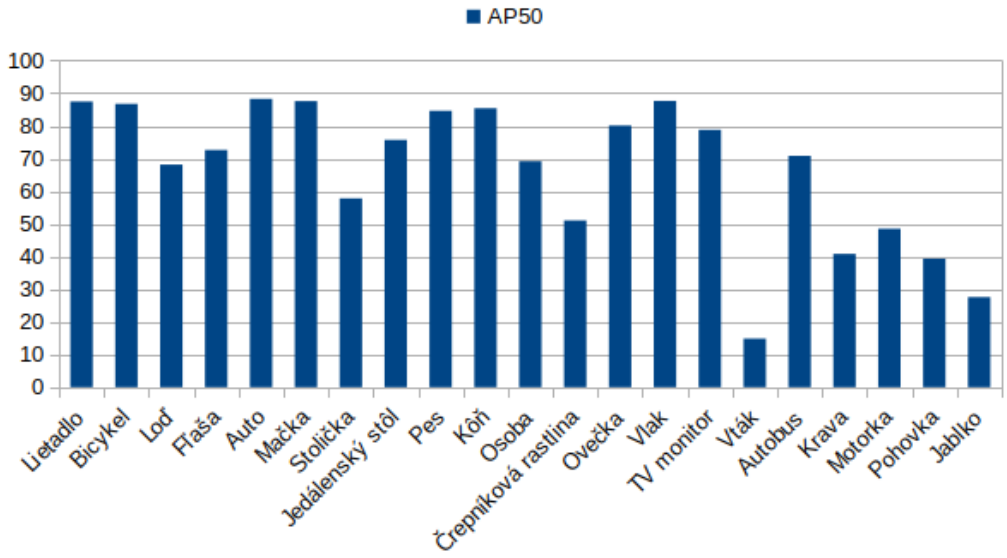
Obr. 4.5: Tabuľka presnosti pre všetky triedy



Obr. 4.6: Celková presnosť pre všetky triedy

Celkový čas tréningu: 1 hodina, 51 minút a 41 sekúnd, na obrázku 4.4 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa zlepšila oproti 1-shot detekcii a už dokážeme rozpoznať každý objekt aspoň s nejakou presnosťou. Na obrázku 4.5 a 4.6 vidíme priemernú presnosť nášho modelu.

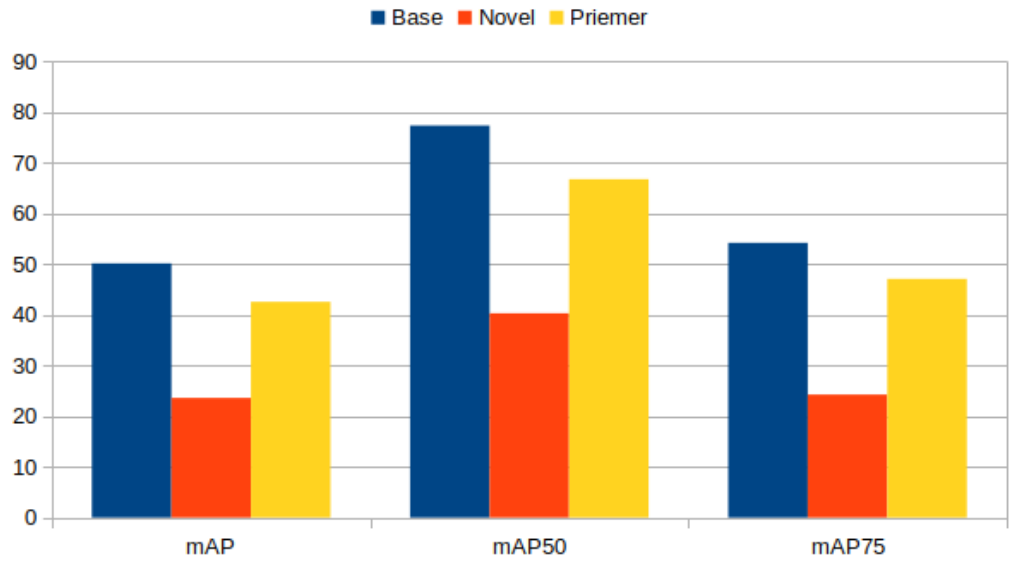
3-shot detekcia



Obr. 4.7: AP50 pre jednotlivé triedy

Presnosť	Base	Novel	Priemer
mAP	50.173	23.598	42.58
mAP50	77.393	40.309	66.798
mAP75	54.235	24.273	47.103

Obr. 4.8: Tabuľka presnosti pre všetky triedy

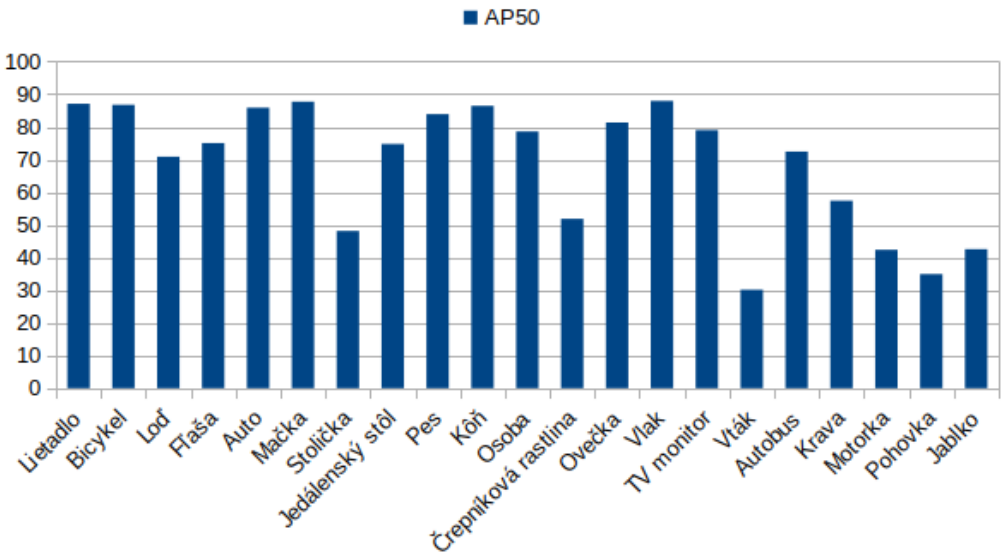


Obr. 4.9: Celková presnosť pre všetky triedy

Celkový čas tréningu: 2 hodiny, 46 minút a 44 sekúnd, na obrázku 4.7 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa opäť zlepšilo, ale pre triedu vták je stále dosť nízka presnosť. Na obrázku 4.8 a 4.9

vidíme priemernú presnosť nášho modelu.

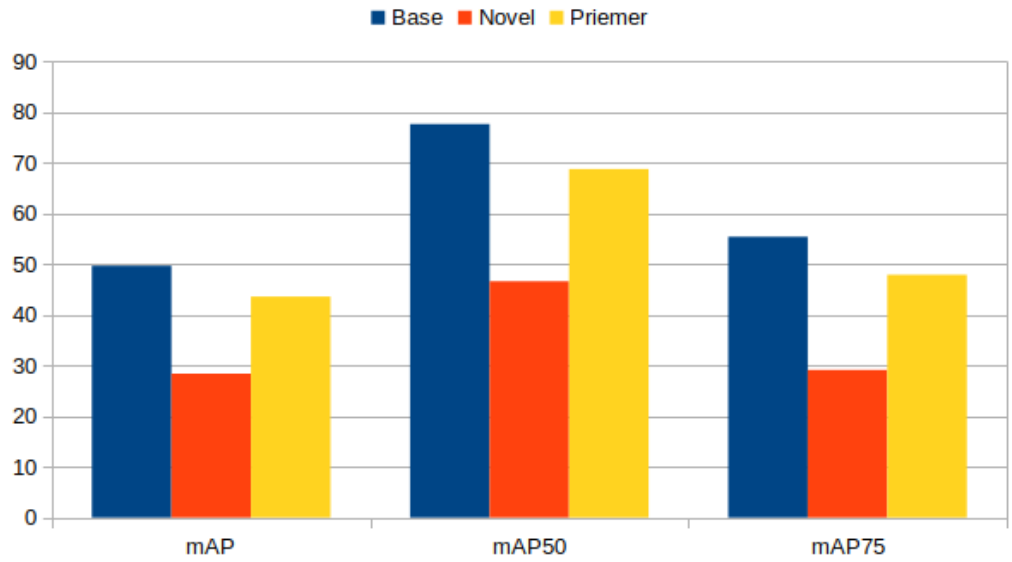
5-shot detekcia



Obr. 4.10: AP50 pre jednotlivé triedy

Presnosť	Base	Novel	Priemer
mAP	49.71	28.395	43.62
mAP50	77.677	46.63	68.806
mAP75	55.442	29.132	47.925

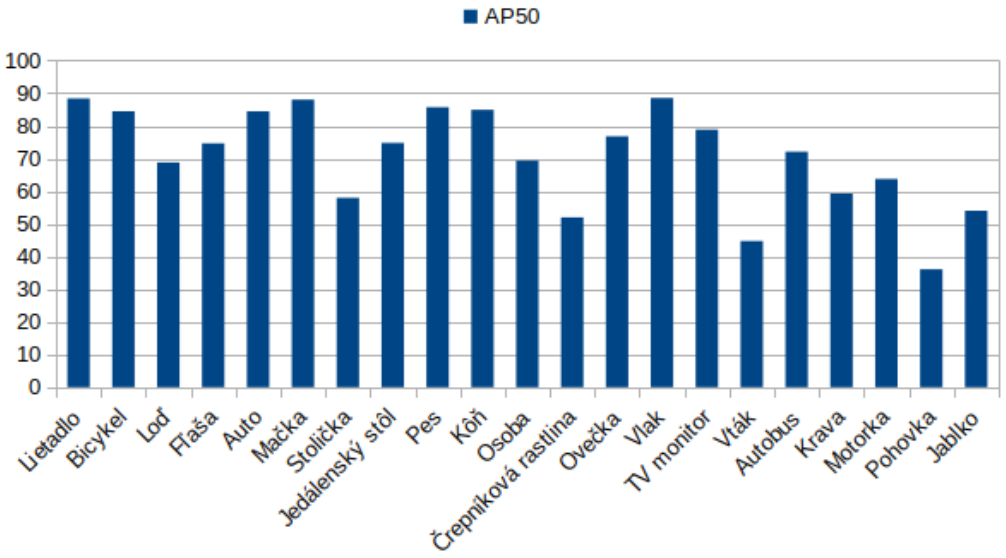
Obr. 4.11: Tabuľka presnosti pre všetky triedy



Obr. 4.12: Celková presnosť pre všetky triedy

Celkový čas tréningu: 4 hodiny, 35 minút a 25 sekúnd. Ako vidíme tréningový čas nám veľmi rastie pri zvyšovaní počtu trénovacích obrázkov, keďže taktiež treba zvyšovať počet iterácií, aby sme dosiahli optimálne výsledky. Na obrázku 4.10 vidíme presnosť AP50 pre jednotlivé triedy, vidíme, že rozloženie presnosti sa zvyšovaním počtu trénovacích obrázkov stále zvyšuje a tentokrát už máme pre každú triedu AP50 nad 30. Na obrázku 4.11 a 4.12 vidíme priemernú presnosť nášho modelu.

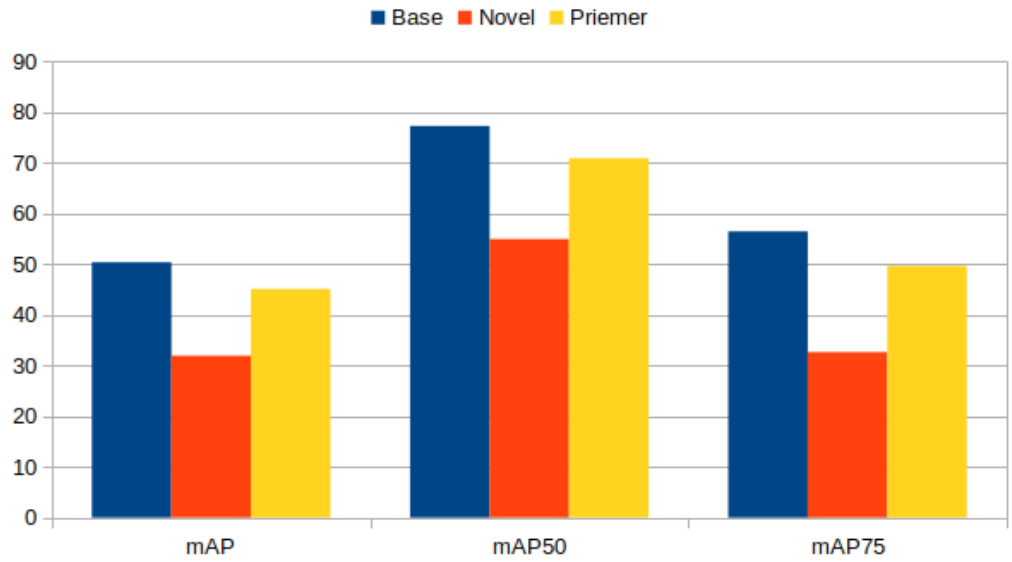
10-shot detekcia



Obr. 4.13: AP50 pre jednotlivé triedy

Presnosť	Base	Novel	Priemer
mAP	50.394	31.946	45.123
mAP50	77.305	54.99	70.93
mAP75	56.49	32.666	49.683

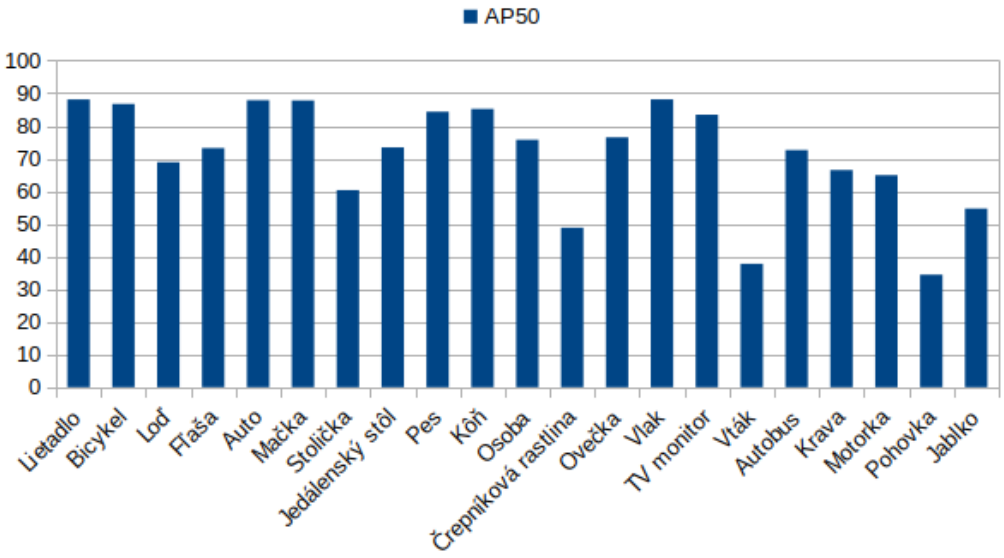
Obr. 4.14: Tabuľka presnosti pre všetky triedy



Obr. 4.15: Celková presnosť pre všetky triedy

Celkový čas tréningu: 9 hodín, 11 minút a 4 sekundy. Na obrázku 4.13 vidíme presnosť AP50 pre jednotlivé triedy. Na obrázku 4.14 a 4.15 vidíme priemernú presnosť nášho modelu.

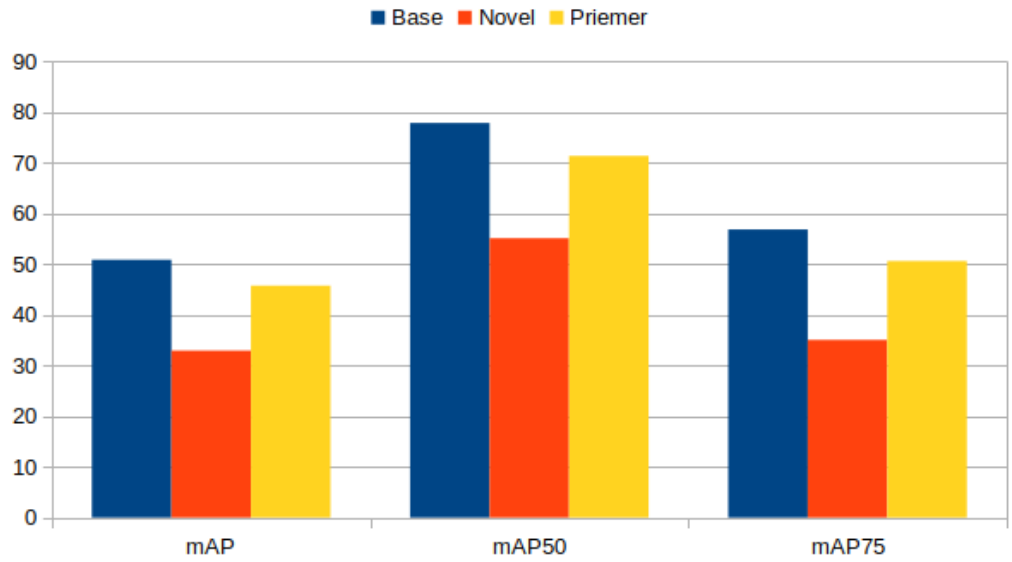
15-shot detekcia



Obr. 4.16: AP50 pre jednotlivé triedy

Presnosť	Base	Novel	Priemer
mAP	50.891	32.951	45.766
mAP50	77.887	55.142	71.388
mAP75	56.88	35.069	50.648

Obr. 4.17: Tabuľka presnosti pre všetky triedy

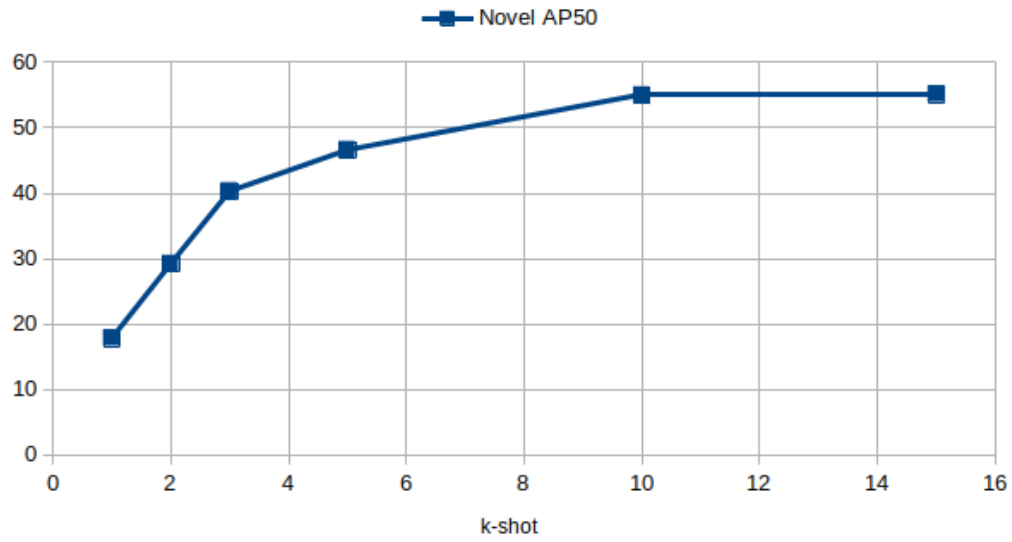


Obr. 4.18: Celková presnosť pre všetky triedy

Celkový čas tréningu: 13 hodín, 56 minút a 37 sekúnd. Na obrázku 4.16 vidíme presnosť AP50 pre jednotlivé triedy. Na obrázku 4.17 a 4.6 vidíme priemernú presnosť nášho modelu. Presnosť nášho modelu sa len minimálne zvýšila oproti 10-shot detekcii a pritom dĺžka tréningu sa zvýšila približne o 50 percent.

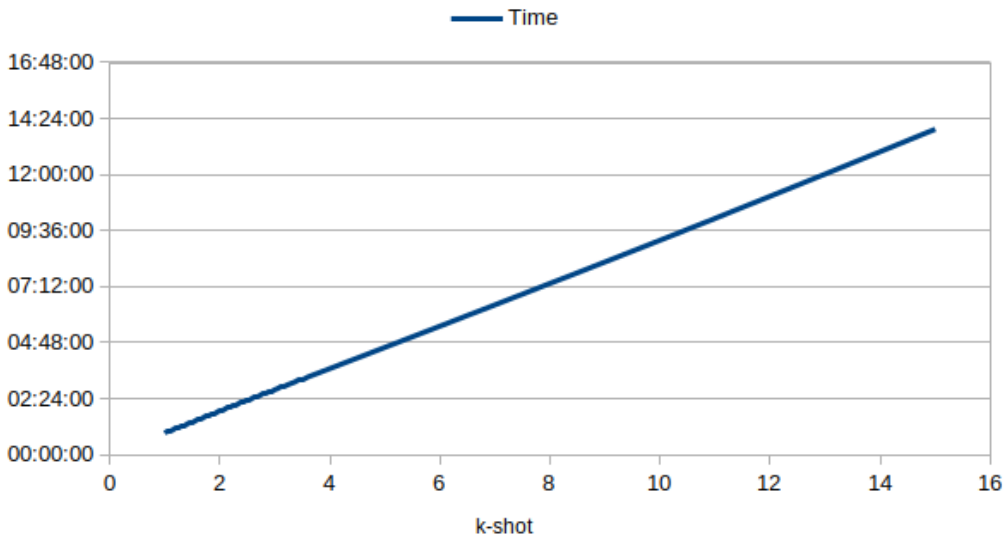
4.2.2 Vyhodnotenie

Terazvyhodnotíme rôzne vlastnosti fine-tuningu vzhľadom k počtu trénovacích obrázkov.



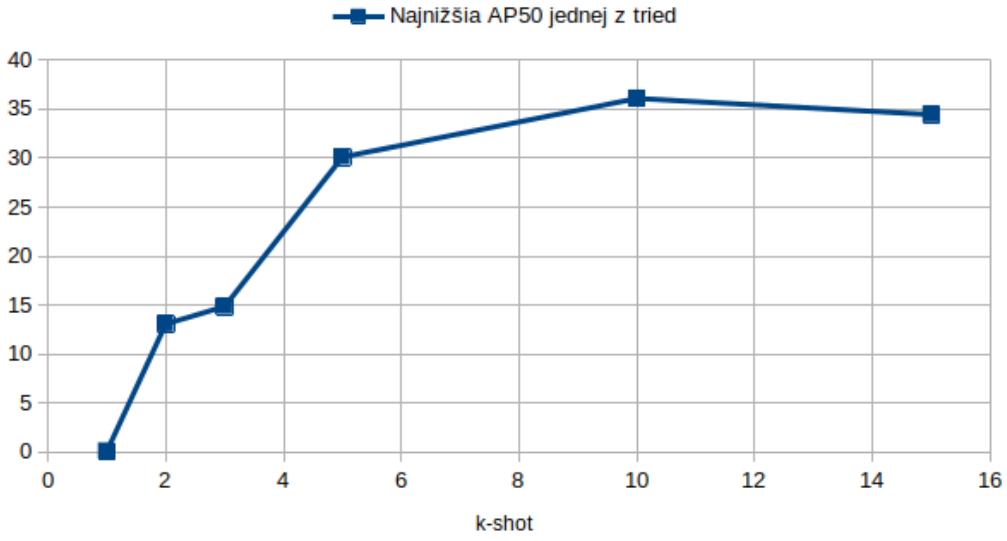
Obr. 4.19: Novel AP50 vzhľadom k počtu trénovacích obrázkov

Na obrázku 4.19 vidíme presnosť novel tried vzhľadom k počtu trénovacíh obrázkov. Vidíme, že presnosť nám stále stúpa, ale po 10tich trénovacích obrázkoch nám presnosť stúpa už iba minimálne. Pri pätnástich trénovacích obrázkoch nám oproti desiatim stúpla presnosť iba o zanedbatelných 0.153.



Obr. 4.20: Čas tréningu vzľadom k počtu trénovacích obrázkov

Na ďaľšom obrázku 4.20 vidíme ako sa nám mení čas tréningu, vzhľadom k počtu trénovacích obrázkov. Vidíme lineárnu funkciu, čas tréningu nám priamoúmerne stúpa s počtom trénovacích obrázkov.



Obr. 4.21: Najnižia AP50 vzhľadom k počtu trénovacích obrázkov

Ďalšia zaujímava metrika na obrázku 4.21 nám ukazuje presnosť tierdy s najmenšou presnosťou vzhľadom k počtu trénovacích obrázkov. Vidíme, že pri 1-shot detekcii jedna z tried má takmer nulovú presnosť, čo nie je úplne ideálne, keď je pre nás dôležité rozpoznávať všetky triedy. Tento graf nám teda zobrazuje garanciu najnižej presnosti pre každú z tried bez ohľadu na priemernú prenosť. Vidíme, že od 5-shot learningu máme celkom solidnú garanciu aspon 30 percent AP50 pre každú triedu. Pri 15-shot detekcii sa nám najnižia presnosť dokonca o trochu zníži oproti 10-shot detekcii, napriek jemnému zvýšeniu priemernej presnosti.

Kapitola 5

Zrýchlenie FSFSOD

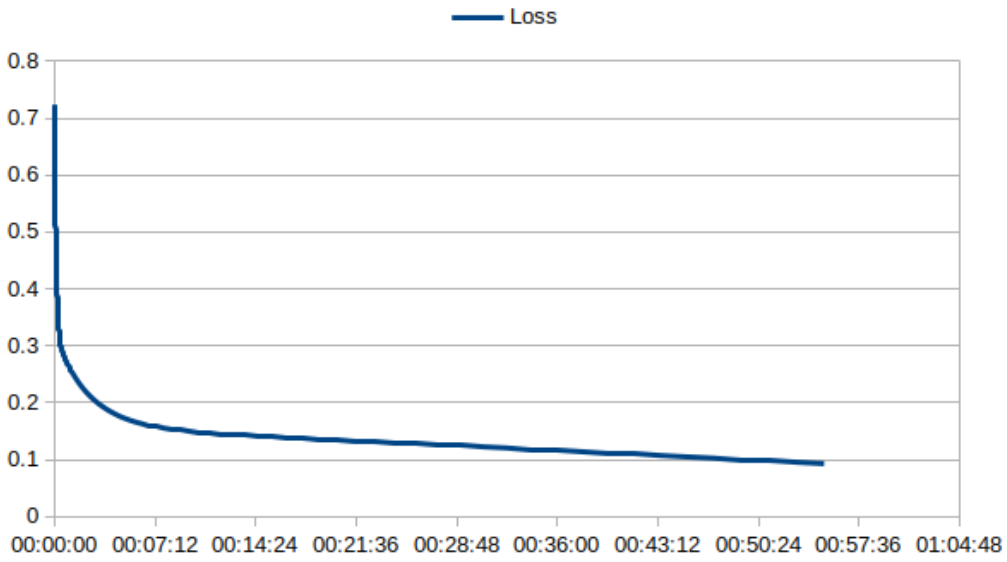
Ako sme videli v predošlej kapitole fine-tuning Frustratingly simple few shot object detection trvá pomere dlho a v žiadnom prípade sa nedá použiť v real-time na učenie nových objektov. Preto sa v tejto kapitole pozrieme na to ako by sa dal zrýchliť.

5.1 Zrýchlenie pomocou zmeny parametrov

Pri pokuse o zrýchlenie fine-tuningu mi ako prvé napadlo zmena trénovacích parametrov. Parametre ktoré majú vplyv na dĺžku tréningu sú batch-size a počet trénovacích iterácii. Pri batch-size sme obmedzený pamäťou našej grafickej karty a maximálny batch size, ktorý zvladne je 2, takže s týmto parametrom nepohneme. Avšak všimol som si, že počas fine-tuningu sa náš model trénuje na začiatku veľmi rýchlo a časom sa veľmi spomaluje.

Najobjektívnejšiu metriku ktorú máme počas tréningu je total loss. Vyskúšame ako sa mení počas 1-shot fine-tuningu. Na začiatku má hodnotu 0.7237 a na konci 0.09278. Behom prvej minúty klesne približne na 0.25. Po piatich minútach ma hodnotu 0.17. Po 20 minútach je na hodnote 0.13 a

ďalších 35minút veľmi pomaly klesá až na finálnu hodnotu 0.09278. Celý priebeh vidíme na obrázku 5.1.

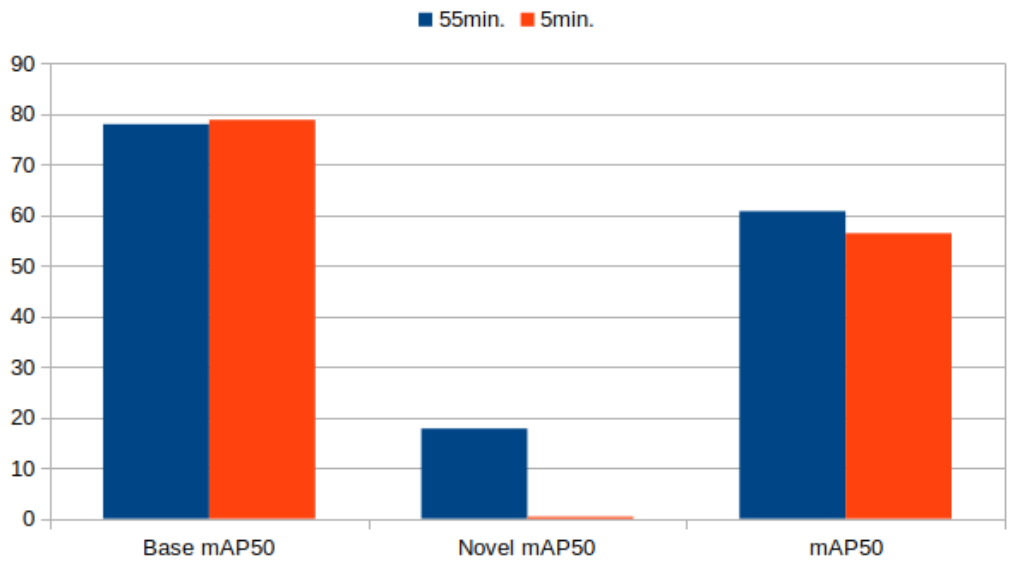


Obr. 5.1: Celková strata počas tréningu

Už po piatich minútach máme náš model podľa tejto metriky výrazne natrénovaný. Pozrime sa teda na ostatné metriky presnosti po 5tich minútach tréningu. Vyskúšame teda najprv znížiť počet iterácií, tak aby tréning trval približne 5 minút a teda znížime počet iterácií z 32 000 na 3 200 a uvidíme ako veľmi nám klesne presnosť pri znížení trénovacieho času o desatinu.

Presnosť	Base	Novel	Priemer
mAP	51.505	0.114	36.822
mAP50	78.865	0.379	56.44
mAP75	57.286	0	40.919

Obr. 5.2: Výsledná tabuľka po 5tich minútach tréningu



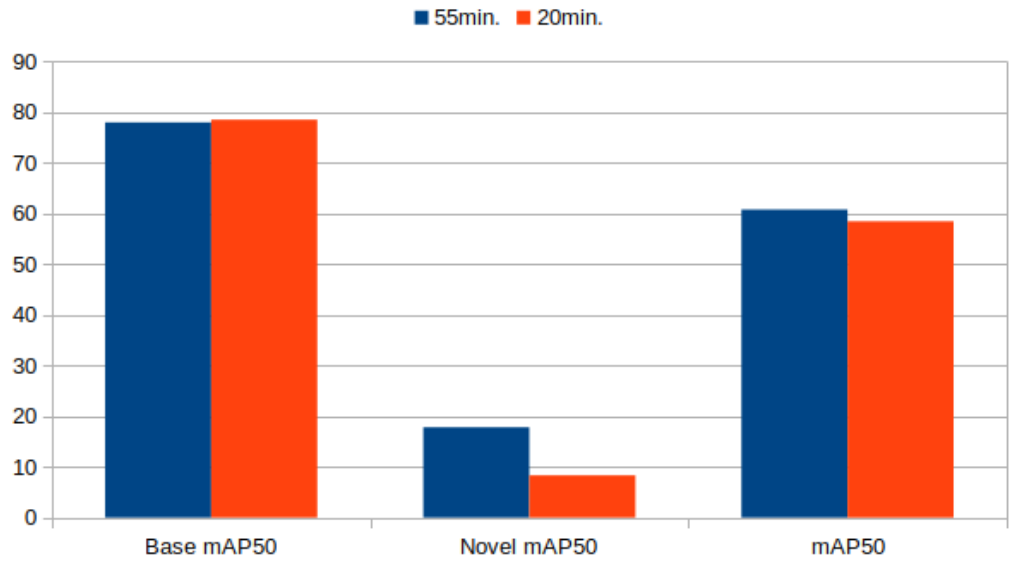
Obr. 5.3: Porovnanie 5min tréningu s plným tréningom

Ako vidíme na obrázkoch 5.2 a 5.3 priemerná presnosť síce ostala cel-

kom vysoká a presnosť pre base classes ostala dokonca vyššia, ale pre nás je najdôležitejšia presnosť pre novel classes a tá má veľmi nízke hodnoty, takže takéto skrátenie tréningu nám prinieslo veľmi zlé výsledky. Vyskúšame teda ešte skrátiť tréning na 20 minút a teda znížime počet iterácii z pôvodných 32 000 na 12 000.

Presnosť	Base	Novel	Priemer
mAP	50.479	4.092	37.226
mAP50	78.541	8.326	58.479
mAP75	56.248	3.701	41.235

Obr. 5.4: Výsledná tabuľka po 20tich minútach tréningu



Obr. 5.5: Porovnanie 20min tréningu s plným tréningom

Na 5.4 a 5.5 vidíme výsledky po 20tich minútach. Priemerná presnosť zvýšila len trochu a vidíme taktiež, že sa nám zvýšila presnosť pre novel classes, ale stále je viac ako o polovicu nižšia ako pri plnom čase tréningu.

5.2 Zrýchlenie pomocou zapamätania si výstupu zo zmrazených vrstiev

Predošlý pokus o zrýchlenie nebol veľmi úspešný, avšak zistil som, že počas tréningu prechádza každý batch vždy celou sieťou, napriek tomu, že pri fine tuningu je takmer celá sieť zmrazená okrem poslednej vrstvy(Box Predictor). Napadlo mi teda vyskúšať prejsť zmrazenou časťou siete pre každý vstup len raz a zapamätať si výstup z tejto časti siete pre každý vstup a následne používať len tento predom vypočítaný výstup zo zmrazenej časti siete pre zvyšok tréningu. Čo by malo výrazne urýchliť prechod sieťou.

5.2.1 Zapamätanie si výstupu z backbone

Ako sme si popísali v 3.kapitole naša sieť sa skladá z 3 hlavných častí: Backbone, RPN a Roi Heads. Drvivú väčšinu našej siete tvorí Backbone, ktorej výstup ide do RPN a Roi Heads. Do Roi Heads ide taktiež výstup z RPN. V našom prípade pri fine-tuningu sú zmrazené všetky vrstvy okrem poslednej - Box Predictor. Keďže vstupom pre Box Predictor je len výstup z Box Head, potrebujeme si zapamätať výstup z Box Head. Keďže, výstupom Box Predicotra je len zoznam s dvomi hodnotami:

1. Class Prediction: pravdepodobnostné rozloženie medzi triedami objektov

2. Bounding Box Regression: posun pôvodných súradníc bounding boxu z RPN

potrebujeme si taktiež zapamätať informácie o návrhoch regiónov, teda výstup z RPN, pre následné vypočítanie straty kvoli tréningu.

Keďže zapamätanie si výstupu pre každý obrázok z Box Head vrstvy, ktorá je súčasťou Roi Heads a výstup návrhov regiónov, ktoré sú výstupom z RPN, je implementačne náročnejšie. Vyskúšame najprv zapametanie si výstupu z Backbone, ktorý aj tak tvori 90% siete a mal by priniesť výrazné zrýchlenie. Ak by nám to neprinieslo očakávané zrýchlenie, nebude mať zmysel posúvať túto myšlienku ďalej a zapamätávať si výstup priamo z Box Head a RPN pre každý obrázok, čo by malo maximalizovať rýchlosť tréningu.

Keďže, počas tréningu používame batch size veľkosti 2 a obrázky, ktoré sa berú náhodne z tréningovej množiny sú rôznej veľkosti a taktiež sú niekoľko krát resizované, tak aby bol zachovaný pomer strán. Tieto dva vybrané obrázky sú prispôsobené na rovnakú veľkosť pomocou paddingu aby mohli byť reprezentované v jednom tenzore, pred tým ako prejdu Backbonom.

Týmto sa nám zapamätávanie výstupov veľmi komplikuje, pretože to aký padding bude pridaný na obrázok zavisí od rozmeru obrázka s ktorým je v batchi. A na to aby sme boli presný museli by sme si pamätať výstup pre každú dvojicu obrázkov zvlášť a to určite nechceme, keďže už pri 1-shot detekcii pri 21 triedach máme 21 obrázkov a každý je 11x resizovaný, čiže 231 obrázkov, teda 231x230 = 53 130 rôznych dvojíc obrázkov a rôznych príznakových máp.

Napadá mi niekoľko možností ako to riešiť:

1. Padding pre všetky obrázky na rovnakú veľkosť

Môžme zistiť aké rozmery má najväčší batch a nastaviť padding pre každý obrázok na tieto rozmery. Všetky obrázky by tak mali rovnaký rozmer a taktiež aj ich príznaková mapa. Čiže by sme mohli vyrátať príznakovú mapu pre každý obrázok vo všetkych veľkostiach a jednoducho by sme potom takéto dve príznakové mapy spojili do batchu. Problém je v tom, že obrázky budú obsahovať zbytočne veľa paddingu, a všetky obrázky budú príliš veľke, čo taktiež veľmi spomalí náš tréning. Pri vyskúšaní tohto postupu som zistil, že tieto veľké obrázky zaberali príliš pamäte a nestačila na to ani moja grafická karta, čiže dosť zlý pokus.

2. Pispôsobiť všetky obrázky na rovnakú veľkosť

Ďalší spôsob, ktorý mi napadol bolo nastaviť všetkým obrázkom rovnakú veľkosť a to spôsobom, že im nastavím veľkosť dlhšej hrany na veľkosť 480, kratšia hrana sa prispôsobí aby sa zachoval pomer strán a následne je obrázok doplnený paddingom tak aby mal rozmer 480x480.

Presnosť	Base	Novel	Priemer
mAP	4.882	0.788	3.712
mAP50	8.875	1.6	6.797
mAP75	4.684	0.874	3.596

Obr. 5.6: Výsledok pre prispôsobené obrázky na rovnakú veľkosť

Presnosť	Base	Novel	Priemer
mAP	4.882	0.788	3.712
mAP50	8.875	1.6	6.797
mAP75	4.684	0.874	3.596

Obr. 5.7: Výsledok pre prispôsobené obrázky na rovnakú veľkosť

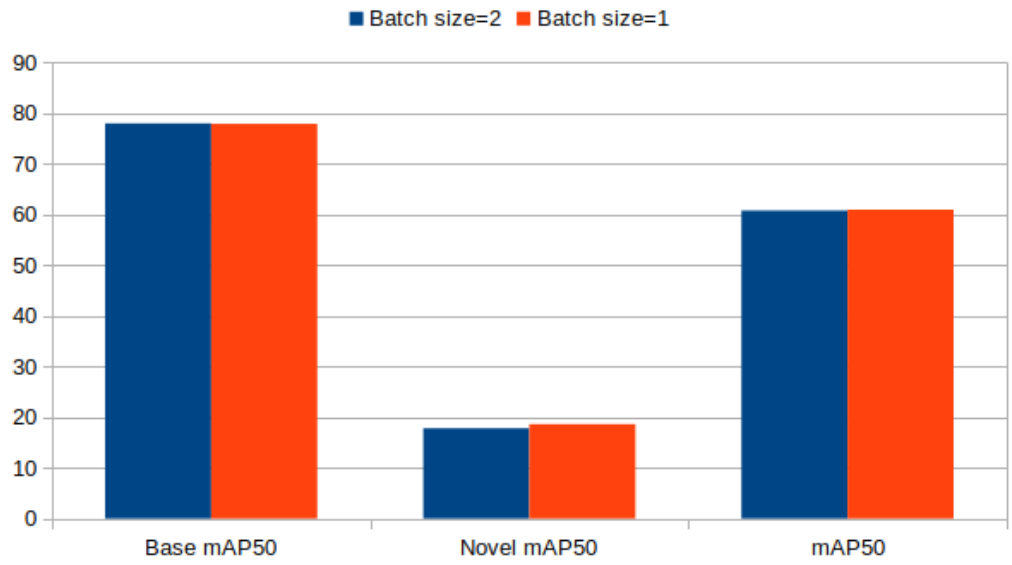
Tréning sa nám síce podarilo výrazne zrýchliť, trval len 13 minút a 27 sekúnd, avšak ako vidíme na obrázkoch 5.6 a 5.7 naša presnosť extrémne klesla.

Takúto nízku presnosť máme hlavne preto, že náš model je testovaný na obrázkoch s veľkosťou dlhšej hrany rovnej 800. Skúsili sme teda zmeniť veľkosť testovacích obrázkov na veľkosť dlhšej hrany rovnej 480. Dosiahli sme tak podobné výsledky ako pri pôvodnom tréningu, avšak tento výsledok

nie je dostatočne dôveryhodný keďže sme si prispôsobili veľkosť testovacích obrázkov ako sme potrebovali a pri reálnom používaní by bol model menej presný.

3. Použiť batch size = 1

Doteraz sme robili tréning pri batch size veľkosti 2, vyskúšame ako bude algoritmus fungovať pri batch size 1. Ak bude dosahovať dobré výsledky, môže nám to uľahčiť zrýchlenie tréningu. Otestujeme to na 1-shot fine-tuningu, pretože prebieha najrýchlejšie. Prispôsobíme teda trénovacie parametre batch size zmenšíme o polovicu na 1, learning rate taktiež znížime o polovicu z 0.000125 na 0.0000625, zdvojnásobime krok z 24 000 na 48 000 a taktiež zdvojnásobime počet iterácií na 64 000.



Obr. 5.8: Porovnanie rôznych batch size

Na obrázku 5.8 vidíme, že tréning pri batch size veľkosti 1 dosahuje veľmi podobné výsledky ako pri tréningu s batch size 2, avšak tréning bol menej stabilný a celková strata(total loss) počas tréningu s batch size 1 skákala z

iterácie na iteráciu s oveľa väčším rozptylom. Čo môže spôsobiť rozptyl vo výsledkoch pri viacerých tréningoch. Každý tréning sa môže podariť inak a pri batch size 1 máme nižšiu konzistenciu.

Toto sa nám ale podarilo vyriešiť pomocou zmeny Checkpointera. Checkpointer zaznamená aktuálny stav modelu a uloží ho do súboru. Predtým sme používali PeriodicCheckpointer, ktorý vytvoril checkpoint periodicky po rovnakom počte iterácií. Počet iterácii po ktorých sa spraví checkpoint je nastaviteľný a mali sme ho nastavený na 500. Použijeme však BestCheckpointer, ktorý spraví checkpoint ak bola total loss nižšia v aktuálnej iterácii ako v poslednom checkpointe. Nastavíme ho aby kontroloval každých 100 iterácií. Následne ako finálny model berieme posledný checkpoint.

Tento tréning s batch size 1 trval dokonca o 4 minúty kratšie. Zrejmä preto, že spojiť 2 obrázky rôznych rozmerov do jedneho batchu je výpočtovo náročnejšie. A keďže osahuje veľmi dobré výsledky, stojí za to ho využiť v tomto princípe.

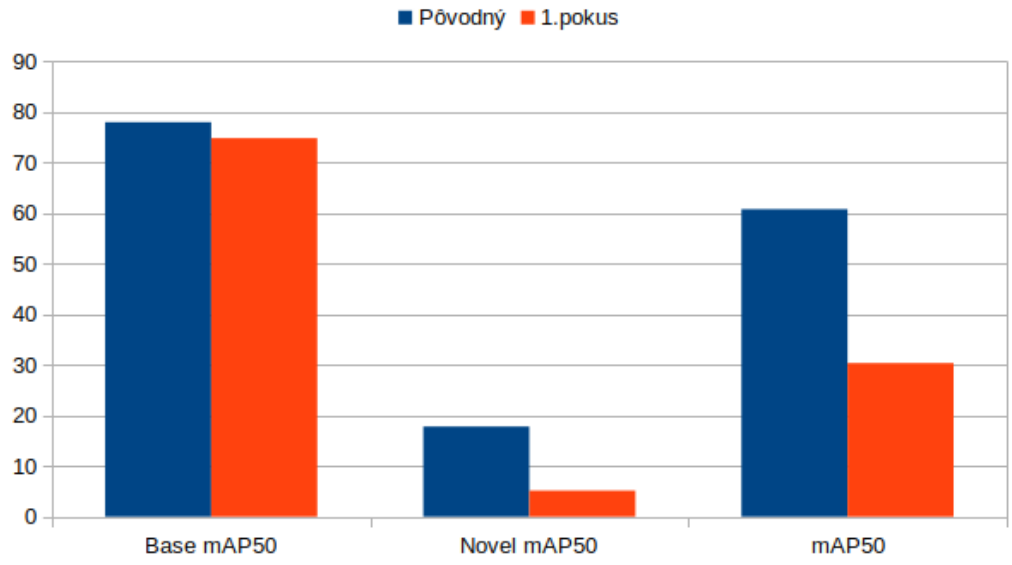
1.pokus

Vyskúšame si teda zapamätať výstup z backbone pre každý obrázok a následne už backbonom prechádzať nebudeme. Uvidíme ako to zýchly náš tréning a aký výsledok dosiahneme.

Čas nášho tréningu výrazne klesol, tréning trval iba 20minút oproti pôvodným 55min. Avšak taktiež klesla naša presnosť ako vidíme na obrázku 5.10 v porovnaní s pôvodným tréningom. Hlavným dôvodom poklesu presnosti bude vynechávanie augmentácií počas tréningu, keďže pre každý obrázok si zapamätáme príznaky iba pre jednu transformáciu toho obrázku. Pôvodne počas sa počas tréningu aplikovali na obrázok rôzne augmentáciu a to zmena veľkosti a taktiež náhodné horizontálne otočenie.

Presnosť	Base	Novel	Priemer
mAP	41.895	1.52	30.359
mAP50	74.852	5.143	54.935
mAP75	41.896	0.347	30.025

Obr. 5.9: Výsledok 1. pokusu



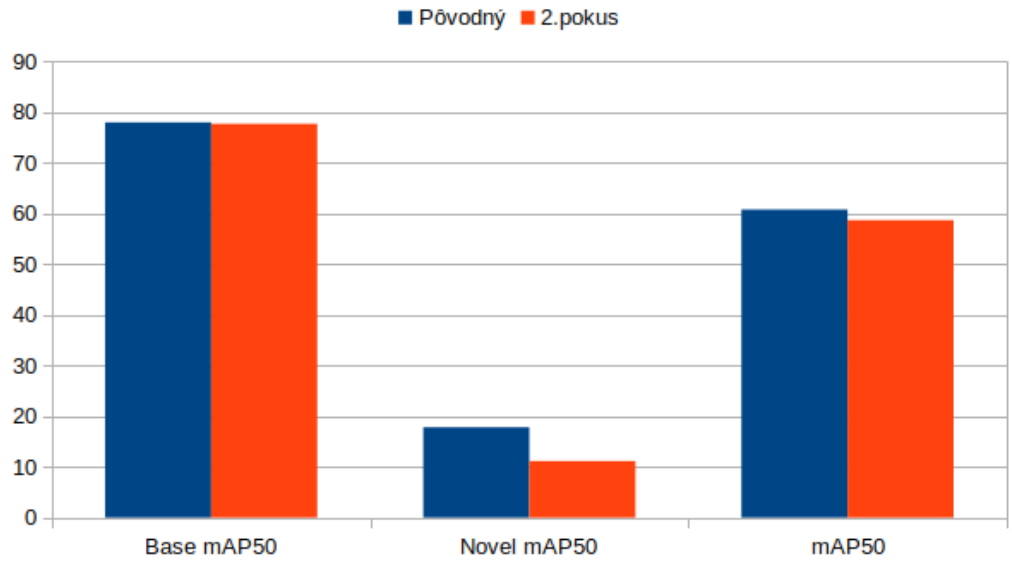
Obr. 5.10: Graf porovnania mAP50 pôvodného algoritmu a 1. pokusu o zrýchlenie

2.pokus

Pre zvýšenie našej presnosti si vyskúšame zapamätať každý obrázok vo všetkých jeho šírkach. Na toto nám však nestačila pamäť v našom gpu. Preto skúsime obrázky resizovať menej krát. Pôvodne sme resizovali každý obrázok 11x na tieto veľkosti kratšej hrany obrázka: 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800. Vyskúšame to zredukovať len na 4 rôzne zmeny veľkosti aby nám stačila naša pamäť v gpu na veľkosti: 480, 608, 704, 800. Tréning trval 20 minút, takže zrýchlenie sme si zachovali a taktiež sa zvýšila naša presnosť. Na obrázku 5.12 vidíme presnosť v porovnaní z predošlími pokusom a taktiež pôvodným nezrýchleným tréningom.

Presnosť	Base	Novel	Priemer
mAP	49.143	4.274	36.323
mAP50	77.716	11.145	58.696
mAP75	54.472	2.65	39.665

Obr. 5.11: Výsledok 2. pokusu



Obr. 5.12: Graf porovnania mAP50 pôvodného algoritmu a 2. pokusu o zrýchlenie

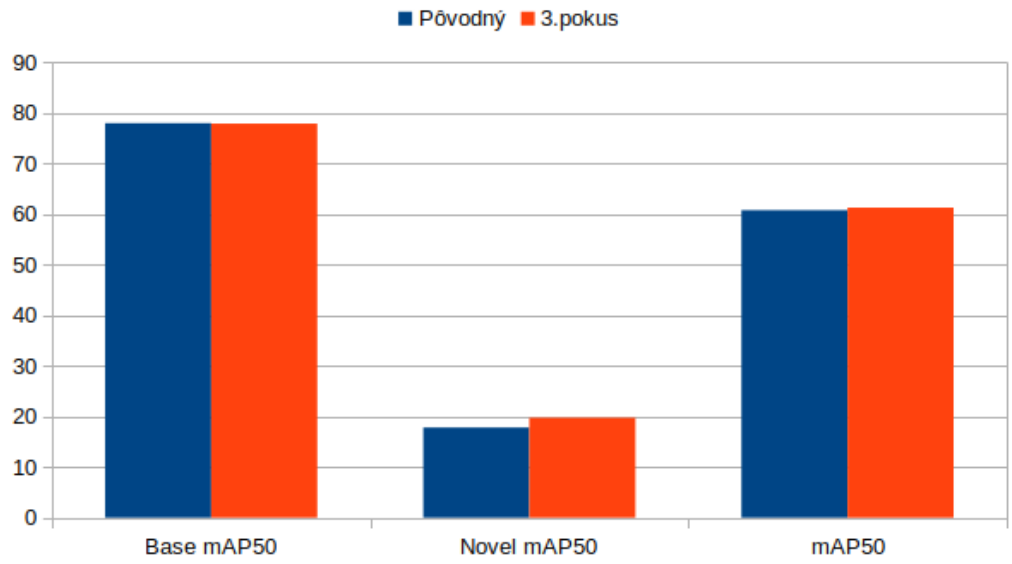
Dosiahli sme solidné výsledky pri 5 násobnom zrýchlení tréningu, avšak stále sme sa nevyrovnali výsledkom pri plnom tréningu a taktiež pri vyššom počte trénovacích obrázkoch nám nebude stačiť pamäť v gpu takže pri nich to nebudeme môcť použiť. Preto sa pokúsime naše príznaky pre každý obrázok v každej veľkosti zapísať do súboru a následne čítať počas tréningu z neho.

3.pokus

Po zapísaní príznakov každého obrázku pre všetky veľkosti do súboru bez náhodného horizontálneho otočenia. Sa náš tréning zvýšil z približne 20minút na 33 minút. Dosiahli sme ale požadované rovnaké výsledky ako pri pôvodnom 55 minútovom tréningu, dokonca o trochu lepšie. Podarilo sa nám teda zrýchliť tréning takmer o polovicu. Na obrázku 5.13 vidíme výsledky tréningu a na obrázku 5.14 porovnanie s pôvodným tréningom.

Presnosť	Base	Novel	Priemer
mAP	49.184	10.262	38.064
mAP50	77.915	19.747	61.296
mAP75	54.58	9.522	41.707

Obr. 5.13: Výsledok 3. pokusu



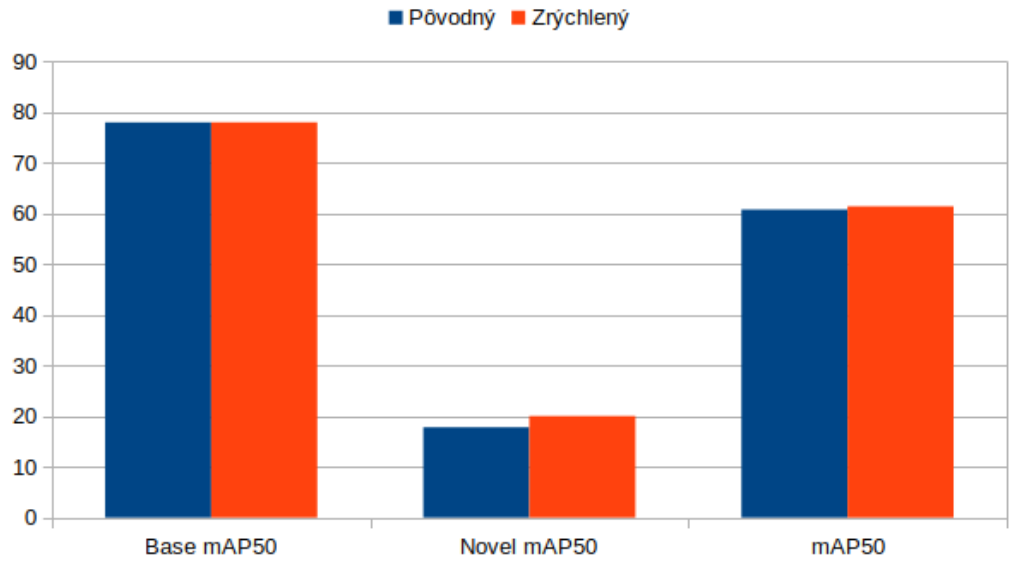
Obr. 5.14: Graf porovnania mAP50 pôvodného algoritmu a 3. pokusu o zrýchlenie

5.2.2 Zapamätanie si výstupu priamo z Box Head

Ako sme videli v predošlej časti, uloženie výstupov z Backbone vrstvy do súborov, pre každý obrázok a pre každú veľkosť zvlášť, nám urýchlilo náš tréning takmer o dvojnásobok a zachovala sa naša presnosť. Pokúsime sa teda tréning ešte zrýchliť uložením si výstupov priamo z Box Head a RPN, keďže tieto dva výstupy sú vstupom pre Box Predictor (posledná vrstva ktorú chceme trénovať).To by malo následne vo zvyšku tréningu výpočtovú zložitosť znížiť a ešte zrýchliť náš tréning. Uvidíme ako veľmi to zrýchli náš tréning v porovnaní z predošlím tréningom.

Presnosť	Base	Novel	Priemer
mAP	49.345	10.218	38.166
mAP50	78.011	20.032	61.446
mAP75	54.54	9.401	41.643

Obr. 5.15: Výsledok zrýchleného algoritmu

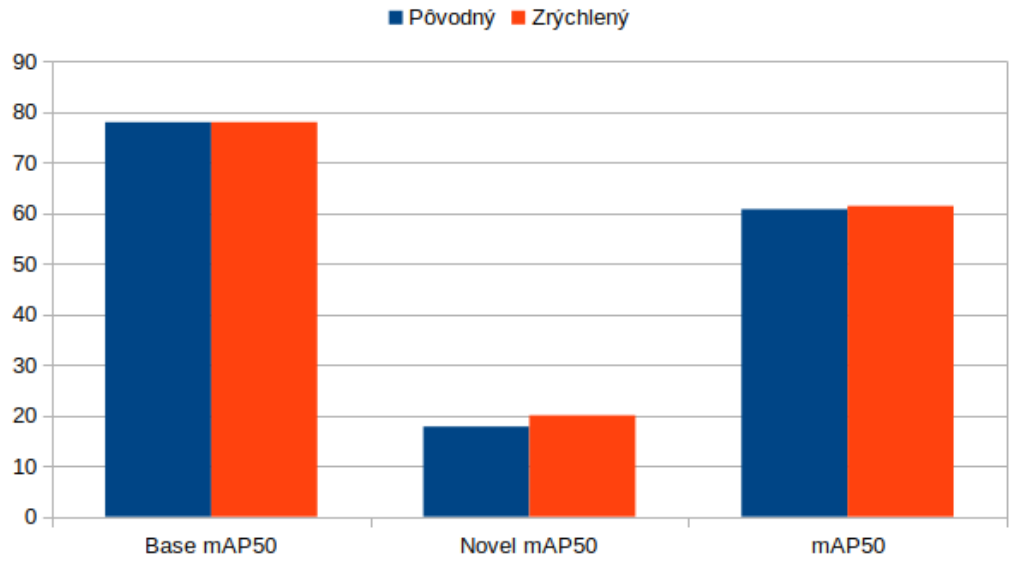


Obr. 5.16: Graf porovnania mAP50 pôvodného a zrýchleného algoritmu

Na obrázku 5.15 vidíme výsledky nášho pokusu a na obrázku 5.16 vidíme porovnanie s pôvodným pomalým algoritmom. Vidíme, že sa nám podarila dosiahnuť rovnaká presnosť. A vyrazne sa nám podarilo zrýchliť tréning celkový čas tréningu bol 6 minút a 51 sekúnd, čo je je takmer 8x rýchlejšie ako pôvodný tréning.

Vyskušal som taktiež spustiť rôzne k-shot fine-tuningy a zaznamenal som odhadovaný čas tréningu. Porovnanie časov tréningov vidíme na obrázku 5.15.

Vyhodnotenie



Obr. 5.17: Porovnanie rýchlostí pôvodného algoritmu a algoritmu po zrýchlení

Na obrázku 5.17 vidíme porovnanie rýchlosti pôvodného algoritmu a algoritmu po zrýchlení pri dosahovaní rovnakej presnosti.

Kapitola 6

Testovanie algoritmu pri zmene počtu tried

V predošlých kapitolách sme si ukázali ako sa mení presnosť a rýchlosť tréningu pri zmene počtu trénovacích obrázkov pre 6 novel tried a taktiež sa nám podarilo poslednú fázu tréningu(fine tuning) 8 násobne zrýchliť. Teraz využijeme náš zrýchlený algoritmus a pozrieme sa na to ako sa mení presnosť a rýchlosť pri zvyšovaní počtu počtu novel tried.

6.1 Fine-tuning na base + novel triedach

Najprv vyskúšame ako sa nám mení presnosť pri zvyšovaní počtu novel tried, pri ponechaní detekcie 15tich base tried, rovnako ako doposial. Neskôr si vyskúšame spraviť fine-tuning len na novel triedach, čo by malo zvýšiť našu presnosť pre novel triedy, s tým, že base triedy nebudeme detegovať.

Pre jednoduchšie a rýchlejšie pridávanie nových tried som napísal zopár scriptov. Popíšem postup ako pridať novú triedu do datasetu:

Anotované obrázky pre nové triedy stiahneme z roboflow vo formate Pas-

cal VOC XML. Pre pridanie novej triedy do datasetu treba rozdeliť anotované obrázky na trénovacie a testovacie. Stačí keď presunieme všetky anotované obrázky do priečinka `datasets/VOC2007/ImageSets/Main/test`. Následne spustíme script `datasets/VOC2007/ImageSets/Main/divideDataset.py`, ktorý rozdelí anotované obrázky do priečinkov train a test. Keďže na fine-tuning nebudeme potrebovať príliš trénovacích obrázkov tak do priečinka train presunieme 30 obrázkov a zvyšok necháme v priečinku test. Potom treba spustiť script, datasets/VOC2007/ImageSets/Main/addNewClass.py, ktorý pridá nové obrázky do textových súborov trainval.txt a test.txt a presunie anotácie do `datasets/VOC2007/Annotations` a obrázky do `datasets/VOC2007/JPEGImages`.

Potom treba spustiť script `datasets/prepare_voc_few_shot.py`, ktorý vytvorí textové súbory pre jednotlivé k-shot tréningy pre každú triedu. Vo formáte `box_kshot_názov_triedy_train.txt`, každý z týchto súborov obsahuje k názvov náhodných obrázkov z danej triedy, ktoré berie z textového suboru trainval.txt a následne sa trénuje a validuje počas fine-tuningu len na nich. Všetky obrázky v testovacej množine sú využité pri testovaní, na vyhodnotenie nášho modelu.

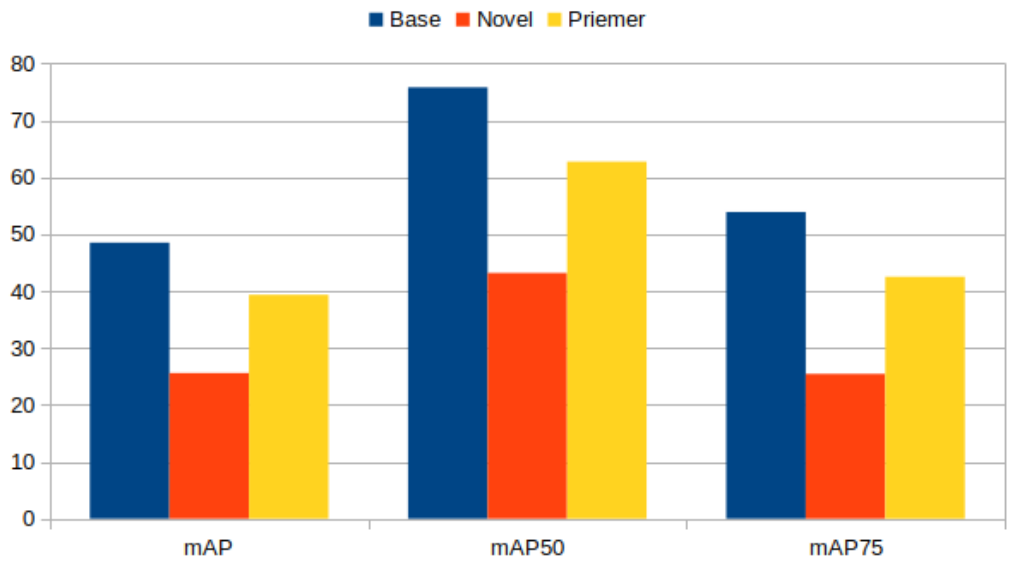
Rozhodol som sa používať pri tomto testovaní 10-shot detekciu, pretože dosahuje vysokú presnosť, rozptyl medzi presnoťou jednotlivých tried je nízky a vieme ju vykonať po našom zrýchlení pomerne rýchlo za cca hodinu.

Nové triedy som vyberal úplne náhodne a keďže závisí presnosť detekcie na ich podobnosti s base triedami, ukážeme si taktiež AP50 pre jednotlivé novel triedy.

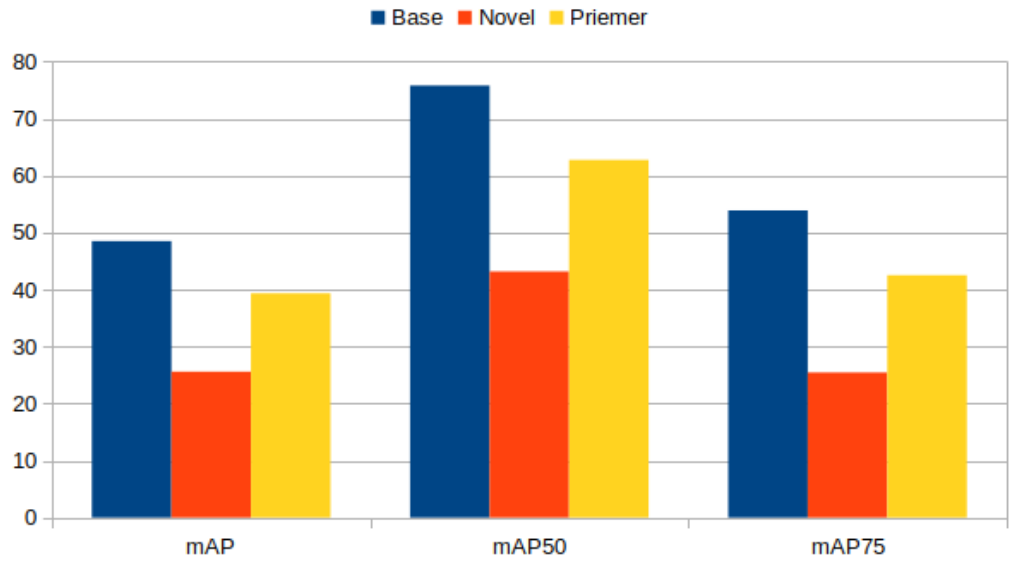
10 novel tried

Presnosť	Base	Novel	Priemer
mAP	48.508	25.594	39.342
mAP50	75.803	43.18	62.754
mAP75	53.901	25.442	42.517

Obr. 6.1: Tabuľka presností pre 10 novel tried



Obr. 6.2: Graf presností pre 10 novel tried



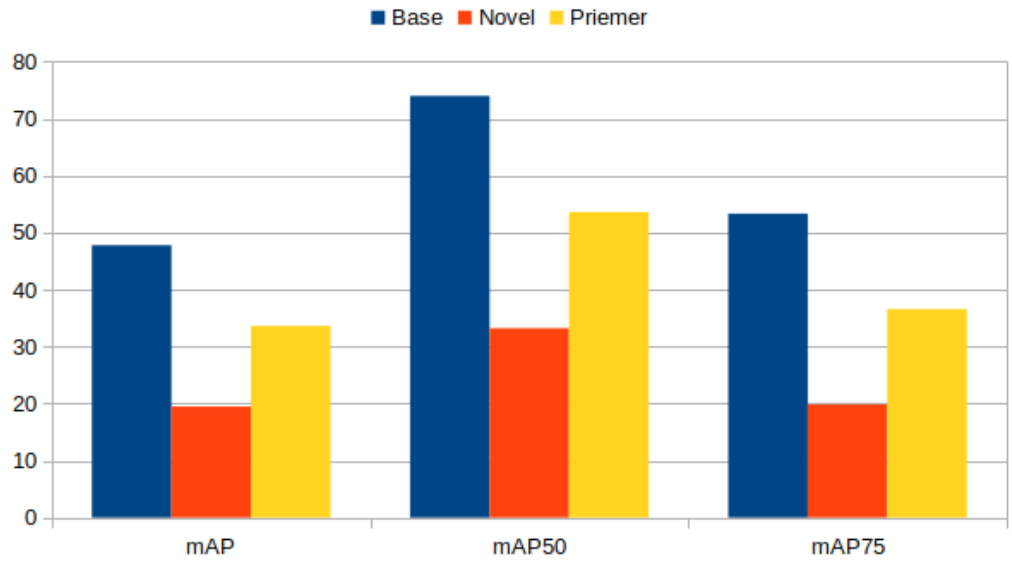
Obr. 6.3: AP50 pre jednotlivé triedy

Tréning trval 1 hodinu, 7 minút a 51 sekúnd. Dĺžka tréningu sa nám o trochu zvýšila oproti 6 novel triedam a taktiež nám trochu klesla presnosť. Ako vidíme na obrázku 6.3 presnosti sú pomerne dobre rozložené a nové triedy si držia solidnú presnosť.

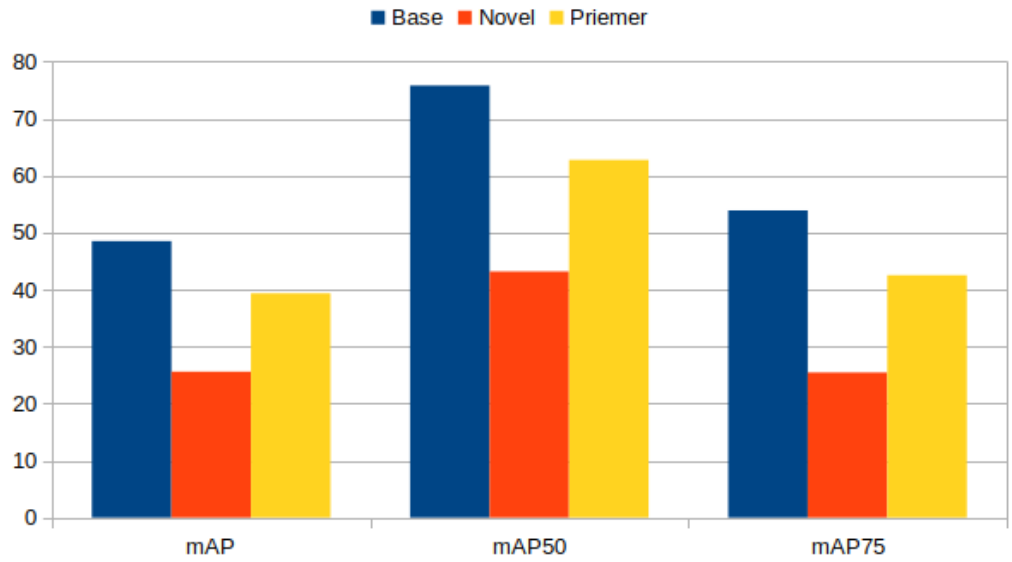
15 novel tried

Presnosť	Base	Novel	Priemer
mAP	47.779	19.471	33.625
mAP50	73.987	33.208	53.597
mAP75	53.33	19.824	36.577

Obr. 6.4: Tabuľka presností pre 15 novel tried



Obr. 6.5: Graf presností pre 15 novel tried



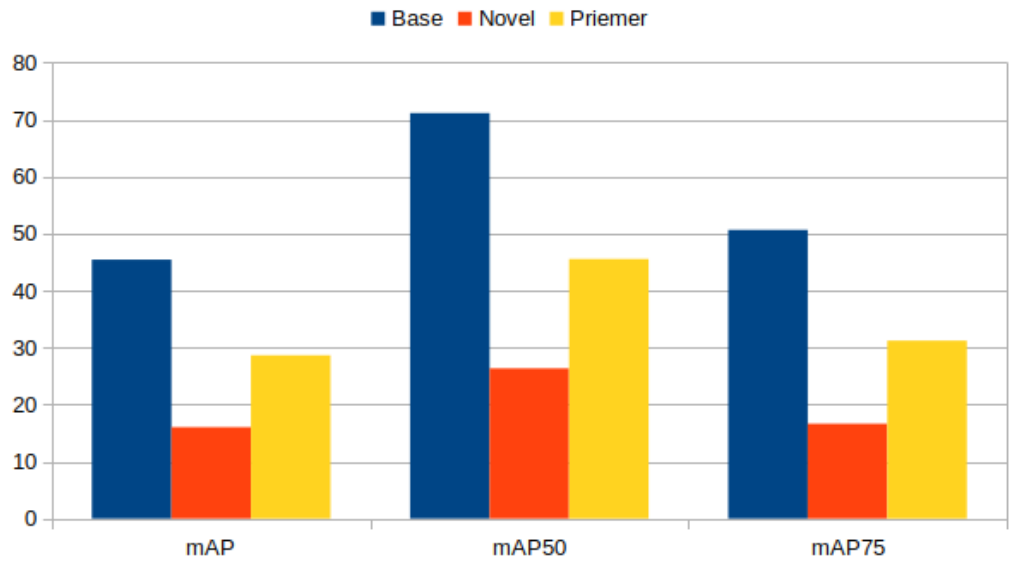
Obr. 6.6: AP50 pre jednotlivé triedy

Tréning trval 1 hodinu, 18 minút a 41 sekúnd. Na obrázkoch 6.4 a 6.5 vidíme presnosť pri 15 novel triedach. Vidíme, že presnosť nám klesla a na obrázku 6.6 vidíme, že sa nám taktiež zvýšila odchílka v presnosti medzi triadami a niektoré triedy majú dosť nízku presnosť, trieda dážnik má takmer nulovú.

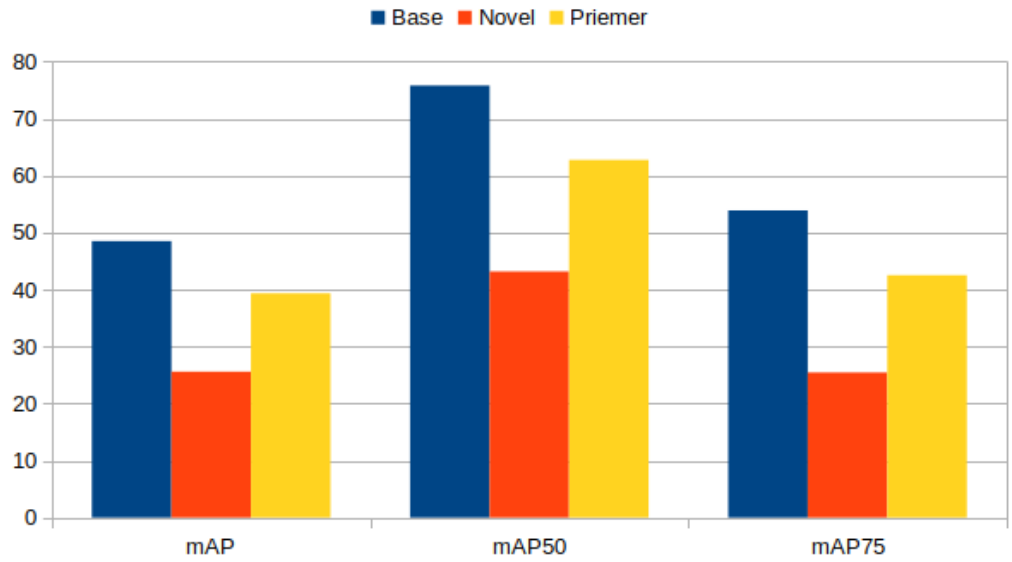
20 novel tried

Presnosť	Base	Novel	Priemer
mAP	45.432	16.031	28.631
mAP50	71.154	26.336	45.544
mAP75	50.659	16.635	31.217

Obr. 6.7: Tabuľka presností pre 20 novel tried



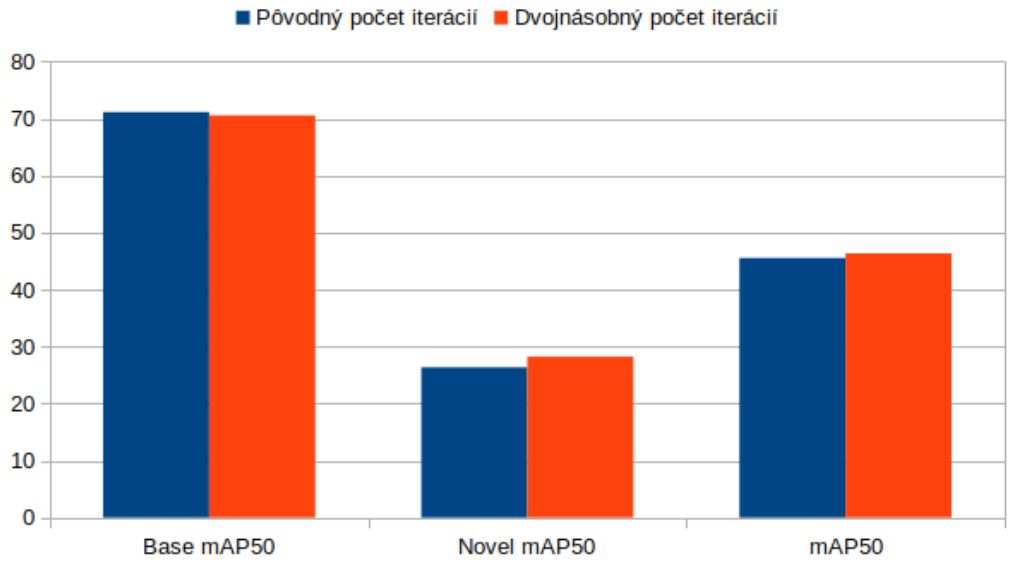
Obr. 6.8: Graf presností pre 20 novel tried



Obr. 6.9: AP50 pre jednotlivé triedy

Tréning trval 1 hodinu, 19 minút a 23 sekúnd. Na obrázkoch 6.7 a 6.8 vidíme presnosťpri 20 novel triedach. Na obrázku 6.9 vidíme presnosť jednotlivých tried. Nové triedy Hotdog, Nôž, Dážnik a Basebalová pálka majú veľmi nízku presnosť AP50 pod 10.

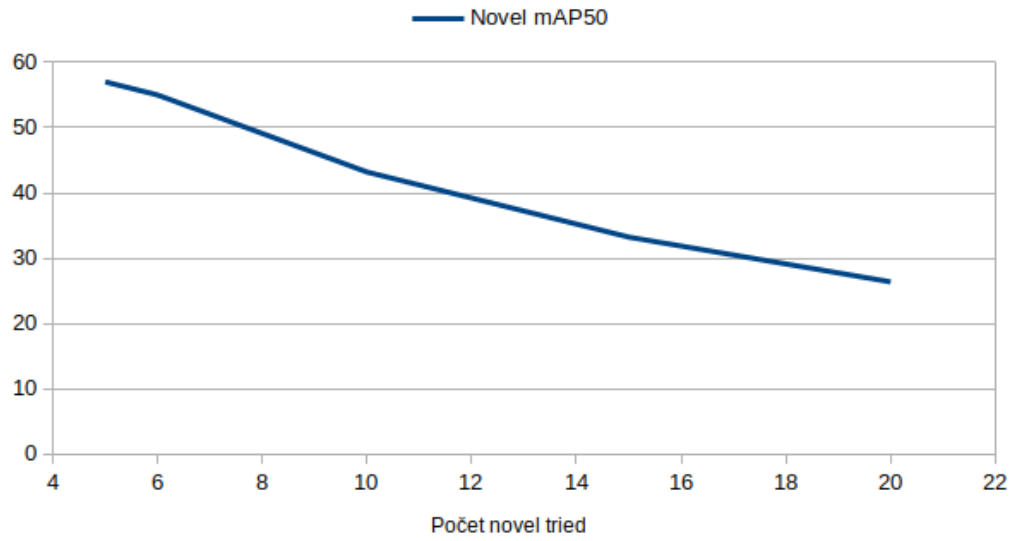
Vidíme, že pri zvyšovaní počtu tried nám presnosť nášho tréningu stále klesá. Vyskúšame zvýšiť počet iterácií tréningu, keďže počet trénovacích obrázkov sa nám zvýšil, skúsime zvýšiť počet iterácií pri 20 novel triedach na dvojnásobok, teda aj čas tréningu bude dvojnásobne dlhý a porovnáme ich presnosť.



Obr. 6.10: Graf presností pre 20 novel tried

Ako vidíme na obrázku 6.10, pri zdvojnásobení počtu iterácií sa nám presnosť trochu zvýšila avšak len minimálne na úkor zdvojnásobenia dĺžky tréningu, takže to nebolo príliš efektívne.

Vyhodnotenie



Obr. 6.11: Presnosť vzhľadom k počtu novel tried

Ako vidíme na obrázku 6.11 presnosť nám pri zvyšovaní počtu novel tried konštantne klesá. Ak by sme chceli udržať našu presnosť pri viacej novel triedach ideálne by bolo natrénovať viacej modelov každý pre detekciu iných tried.

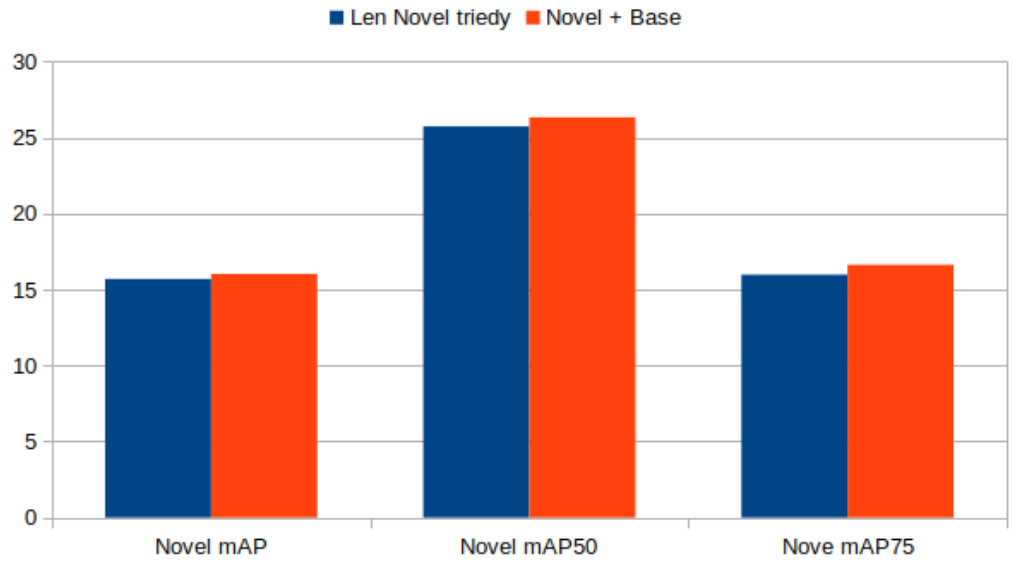
6.2 Fine-tuning len na novel triedach

Ak nepotrebujeme detegovať base triedy, môžu byť využité len pri 1.fáze tréningu (base tréning), na predtrénovanie celej siete. Následný fine-tuning môžme robiť len na novel triedach, čo by malo zvýšiť presnosť ich detekcie.

Vyskúšame spraviť 10-shot fine-tuning na 20 novel triedach, bez base tried a porovnáme presnosť detekcie pri detekcii 20tich novel tried spolu s 15 base triedami.

Presnosť	Len Novel triedy	Novel + Base
Novel mAP	15.696	16.031
Novel mAP50	25.74	26.336
Nove mAP75	15.978	16.635

Obr. 6.12: Tabuľka porovnania presnosti pre fine-tuning bez a s base triedami



Obr. 6.13: Graf porovnania presnosti pre fine-tuning bez a s base triedami

Ako vidíme na obrázkoch 6.12 a 6.13 fine-tuning čisto na novel triedach nenaplnil naše očakávania a presnosť detekcie novel tried sa dokonca znížila.

Kapitola 7

Záver

Literatúra

[1] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[2] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[4] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[5] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-

cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[13] Xin Wang, Thomas E Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. *arXiv preprint arXiv:2003.06957*, 2020.

[14] Lorenzo Natale Elisa Maiettini, Vadim Tikhanoff. Weakly-supervised object detection learning through human-robot interaction. *IEEE*, (1):8, 2021.

[15] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.

[16] Guangxing Han, Shiyuan Huang, Jiawei Ma, Yicheng He, and Shih-Fu Chang. Meta faster r-cnn: Towards accurate few-shot object detection with attentive feature alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 780–789, 2022.

Zoznam obrázkov